Chapter 2: Array, Functions and String (14 marks)

Array - declaring an Array, Initializing an Array, defining an Array elements, Looping an Array, Adding an Array element.
sorting an Array element, Combining an Array elements into a String, changing elements of an Array, Objects as associative Arrays.
Function — defining a function, writing a function, adding an arguments, scope of variable and arguments,
Calling a function — calling a function with or without an argument, calling function from HTML.
Function calling another function. Returning a value from a function.
String - Manipulate a string, joining a string retrieving a character from given position,
retrieving a_ position of character in a string, dividing text, copying a sub string, converting string to number and numbers to string,
changing the case of string, finding a Unicode of a character-charCodeAt(), fromCharCode().

ARRAYS

Declaring an Array

By using Array Constructor

Following JS code line is used to declare array object.

```
var fruit = new Array()
```

what will happens when the browser executes above declaration statement.

First, the browser finds an empty spot in memory and then reserves it for the array. The browser then associates that memory location with the word `fruit`.

Next the browser creates an instance of the array object. In order to create the instance, we need to use the new operator and the `Array()` constructor. Think of a constructor as a special method of an object that creates the instance.

The assignment operator (=) tells the browser to store the new instance of the array object at the location that is associated with `fruit`.

Initializing an Array

To initializing an array, place the value within the parentheses of the `Array()` constructor. The following example initializes the products array with the value 'Soda', which is assigned to the fi rst element of this array:

```
var fruit = new Array(‘Apple')
```

In the real world, an array usually has more than one array element, with each element having its own value. Therefore, you’ll fi nd yourself having to initialize the array with more than one value. Here’s how this is done:

```
var fruit = new Array('Apple', 'Kiwi', 'Pineapple', 'Watermelon')
```

Defining Array Elements

I Javascript array, **to access** array elements individually indexing can be used. Indexing in JS array starts from 0 and increments till length minus one of array.

fruit	Apple	Kiwi	Pineapple	Watermelon
	0	1	2	3

```
console.log(fruit[0])  //will display first element value from array fruit.
```

To assign a fruit name to an array element, you must specify the name of the array followed by the index of the array element. The index must be enclosed within square brackets.

```
fruit[4] = ‘Orange'
```

Looping the Array

Suppose you need to display all the array elements on a document. Remember that you place the information you want displayed between the parentheses of the `document.write()` method. If you use four variables—one for each fruit—you'll have to write the `document.write()` method in four different statements within your JavaScript. But if you use an array, you'll have to write the `document.write()` method in only one statement. Here's how this is done:

```
for (var i = 0; i < fruit.length; i++)

{  document.write(fruit[i])  }
```

The loop begins by initializing variable i to the value 0. Remember that the value of the length property is 4 because there are four elements in the fruit array. Since the value of i is less than 4, the browser executes the statement within the loop.

Notice the index of the array element is `i`. The browser replaces the `i` with the current value of the variable `i`. So initial value of `I`, It is `0`, according to the JavaScript. Therefore, the browser writes the value of array element `0`.

The browser returns to the top of the for loop and increments (`i++`) the value of `i`, and then next element of fruit array will be displayed, this will be continued till `i` become equal to length of array.

Adding an Array Element

At last by using index

Problem while we are going to new element at last in array is that “How do you know what index to assign to the new array element?”.

The solution is to use the length property of the array, as illustrated below:

```
fruit[fruit.length] = 'Mango'
```

By using push() method

push() method used to add an array element to array at last. push() method will return length of array after adding new element. We can multiple elements by seperating them using comma.

Syntax: `array-object . push(element)`

Example:

```
fruit . push(‘carrot’)  //will add ‘carrot’ element at last of fruit array
```

Adding an Array Element continue..

By using unshift() method

push() method used to add an array element to array at last while unshift() method is used to add element at first. unshift() will return length of array after adding new element. We can add multiple elements by separating them using comma as shown in example.

Syntax:

```
array-object.unshift(element)
```

Example:

```
A = [23 , 'asd' ,34]

console.log(A.unshift(45,56)) // will return length so output is 5

console.log(A) // [ 45, 56, 23, 'asd', 34 ]
```

Sorting an Array element

array.sort() is called on the array we want to sort. However, by default, it considers the elements of the array as strings; “100” is lower than “50” because “1” comes before “5”.

To avoid this, an optional argument, which is a user-defined compare function, can be passed to this function.

Compare Function(a, b)

In case we pass this optional parameter, the sort() method takes the value at index 0 as a and every other value one by one as b to pass on to this compare function. Sorting is then done based on its return value:

1. return value < 0 : a is placed before b.
2. return value > 0 : a is placed after b.
3. return value = 0 : Keeps the same order as in the original array.

Array sort example.

```
// Without compare function
          var A = [50, 100, 2, 0];
          console.log("Without compare function:");
          A.sort();
          console.log(A);
Output:   Without compare function:
          [ 0, 100, 2, 50 ]
```

```
// With a compare function – Defined in sort call statement
           var A = [50, 100, 2, 0];
           console.log("With a compare function:");
           A.sort(function(a, b){ return a - b;});
           console.log(A);
 Output:   With a compare function:
           [ 0, 2, 50, 100 ]
```

```
// With a compare function – Defined out of sort call statement
          var A = [50, 100, 2, 0];
          console.log("With a compare function:");
          Function srt(a, b)
          { return a - b;
          }
          A.sort(srt); //will first element as a rest as b
          console.log(A);
 Output:  With a compare function:
          [ 0, 2, 50, 100 ]
```

Array sort example.

```
// JAVASCRIPT PROGRAM TO SORT ARRAY ELEMENTS ACCORDING TO LENGTH OF ELEMENTS

          fruit = ['Apple', 'Kiwi', 'Pineapple', 'Watermelon']
          function srt(a, b)
                    {
                    return a.length - b.length;
                    }
         fruit.sort(srt);
         console.log(fruit);
```

Output: `[ 'Kiwi', 'Apple', 'Pineapple', 'Watermelon' ]`

Combining an Array elements into a String

By toString() method

The toString() method returns a string with array values separated by commas. The toString() method does not change the original array.

Syntax:

```
array-object.toString()
```

Example:

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
str = fruits.toString();
console.log(str)
```

Output:

Banana,Orange,Apple,Mango

Combining an Array elements into a String continue…

By join() method

The join() method returns an array as a string. The join() method does not change the original array.

Any separator can be specified. The default is comma.

Syntax:

```
array-object.join(separator)
```

Example:

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
str1 = fruits.join();
console.log(str1)
str2 = fruits.join('*');
console.log(str2)
```

Output:

```
Banana,Orange,Apple,Mango
Banana*Orange*Apple*Mango
```

Changing Elements of the Array

Most of us are familiar with to-do lists. New tasks are placed at the bottom of the list and eventually move to the top when all the other tasks ahead of it are completed and removed from the list. An array can be used as a to-do list.

Examples:

```
var ToDoList = new Array()

ToDoList[0] = "Book the Waldorf for your birthday party."

ToDoList[1] = "Give the Donald a call and invite him to your party."

ToDoList[2] = "Leave word at the White House that you won't be available for dinner."
```

Suppose that you have booked the Waldorf, so you need to remove the fi rst task from the list. You do this by calling the shift() method of the array object. The shift() method removes and returns the first element of the array and then moves the other tasks up on the list. Here's how to call the shift() method:

```
var task = ToDoList.shift()

Here's the ToDoList array after the shift() method is called:

ToDoList[0] = "Give the Donald a call and invite him to your party."

ToDoList[1] = "Leave word at the White House that you won't be available for dinner."
```

FUNCTIONS

Defining Functions

A function must be defi ned before it can be called in a JavaScript statement.

A function defi nition consists of four parts: the name, parentheses, a code block, and an optional return keyword.

The **function name** is the name that you've assigned the function. The name must be

The name cannot

- Begin with a digit
- Be a keyword
- Be a reserved word

Parentheses are placed to the right of the function name at the top of the function definition.

The **code block** is the part of the function defi nition where you insert JavaScript statements that are executed when the function is called by another part of your JavaScript application.

The **return** keyword tells the browser to return a value from the function definition to the statement that called the function.

Writing a Function Definition

To write function definition use *function* keyword.

Syntax

```
function function_name (arguments list)
        {
        //Statements to
        //perform
        //function task
        return
        }
```

Example

```
function add (n1,n2)
        {
        var sum
        sum = n1 + n2
        return sum
        }
```

In example we have given function name as *add* and there are two arguments are passed the first one is n1 the second one is n2. The sum of n1 and n2 return to the function call statement.

The Scope of Variables and Arguments

Local Variable

A variable can be declared within a function, such as the **`sum`** variable in the **`add()`** function. This is called a local variable, because the variable is local to the function. Other parts of your JavaScript don't know that the local variable exists because it's not available outside the function.

Global variable

A variable can be declared outside a function. Such a variable is called a global variable because it is available to all parts of your JavaScript - that is, statements within any function and statements outside the function can use a global variable.

Now consider following examples:

The Scope of Variables and Arguments examples ….

Example:

```
var sum = 20
function num_print()
        {
         sum =90
         console.log(sum) //will print 90
        }
num_print()
console.log(sum); //will print 90
```

Output:

```
90
90
```

Since sum variable declared globally using var, so inside function sum (global variable accessed).

Example:

```
var sum = 20
function num_print()
        {
         let sum =90
         console.log(sum) //will print 90
        }
num_print()
console.log(sum); //will print 20
```

Output:

```
90
20
```

Since sum variable declared globally using var, so inside function sum (global variable accessed) but if we want to declare sum as local we have to use let keyword.

Calling a Function

A function is called by using the function name followed by parentheses. If the function has arguments, values for each argument are placed within the parentheses.

You must place these values in the same order that the arguments are listed in the function definition. A comma must separate each value.

```
add()  //add function called without argument

add(200,30)  //add function called with two arguments
```

Calling a Function from HTML

A function can be called from HTML code on your web page. Typically, a function will be called in response to an event, such as when the web page is loaded or unloaded by the browser.

You call the function from HTML code nearly the same way as the function is called from within a JavaScript, except in HTML code you assign the function call as a value of an HTML tag attribute. Let's say that you want to call a function when the browser loads the web page.

what you'd write in the <body> tag of the web page:

```
<body onload = "WelcomeText()">
```

Calling a Function from HTML example

```
<html >
<head>
        <title>Calling a function from HTML</title>
        <script language="Javascript" type="text/javascript">
                function WelcomeText()
                {  alert('Happy to see you.')
                }
                function ByeMessage()
                {  alert('Bye.')
                }
        </script>
 </head>
        <body onload="WelcomeText()"   onunload="ByeMessage()">
        </body>
</html>
```

Functions Calling Another Function example

Functions, as you learned earlier in this chapter, can be called from any JavaScript or from HTML code on a web page itself. Consider following example in which count_vowel called inside function get_str.

Vowels.html

```
<html>
    <body>
        <label for="">String</label>
        <input type="text" id="str">
        <label for="">No of Vowels</label>
        <input type="text" id="cnt">
        <button onclick="get_str()">COUNT VOWELS</button>
    </body>
    <script src="Vowels.js"> </script>
</html>
```

Vowels.js

```
function get_str() {
    s = document.getElementById("str").value;
    c = count_vowels(s); //function count_vowels called inside another function
    document.getElementById("cnt").value = c
}
function count_vowels(s) {
    c = 0
    for (i in s) {
                if (s[i] =='a' || s[i] =='e' || s[i] =='i' || s[i] =='o' || s[i] =='u')
                        {
                        c = c + 1
                        }
                }
return c
}
```

STIRNGS

Joining Strings

When you concatenate a string, a new string from two strings by placing a copy of the second string behind a copy of the first string. The new string contains all the characters from both the first and second strings.

You use the concatenation operator (+) to concatenate two strings, as shown here (note that, in this context, + is the concatenation operator, not the addition operator):

```
NewString = FirstString + SecondString
```

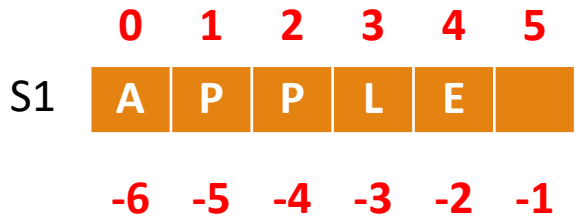
Suppose you needed to display a customer's full name on the screen. However, the customer's name is stored in the database as two data elements called FirstName and LastName.

You'll need to concatenate the first name and the last name into a new string and then display the new string on the screen. This is illustrated in the next example

```
<html>
<body>    <input type=“text” name=“fname”/>
          <input type=“text” name=“lname”/>
</body>
<script language="Javascript" type="text/javascript">
          Fn = document.getElementById(‘fname’).value
          Ln = document.getElementById(‘lname’).value
          var newString = Fn + Ln
          alert(newString)
</script>
</html>
```

Retrieving a character from given position

You know that a string is an array of characters. Each character in a string is an array element that can be identified by its index. Take a look at the following example to see how this works



Since individual character in a string is identified by index, we can retrieve a character by using index.

```
e.g. console.log(S1[3]) //will print character L
```

charAt() method :

The **charAt()** method requires one argument, which is the index of the character that you want to copy. The following statement illustrates how to use the charAt() method:

```
Synatx:    var SingleCharacter = NameOfStringObject.charAt(index)

Example:   var c = S1.charAt(3)     //will print character L
```

Suppose you wanted to use the last four digits of a person's Aadhar number for the person's ID. You can copy these digits to a new string, but you need to know the index of the first of the four digits.

We can use the length value to calculate the index of the character that we want to use. Here how this is done:

```
var AadharNumber = '123456789'

var IndexOf_4_Character = AadharNumber.length – 4

var IndexOf_3_Character = AadharNumber.length - 3

var IndexOf_2_Character = AadharNumber.length - 2

var IndexOf_1_Character = AadharNumber.length - 1
```

Retrieving a position of character in a string

indexOf() method:

indexOf() method return the position of the first occurrence of a value in a string.

If the value is not found in a given string indexOf() method will return -1.

searchValue parameter is required and indicate the string to be searched.

start parameter is optional and indicate position, from where to start search in a main string.

Syntax:

```
String.indexOf(searchValue , start)
```

Example:

```
S1 = “POLYTECHNIC”
pos = S1.indexOf(‘TECH’)
console.log(pos)    // out put will be 4
```

Dividing Text

Look at the following code segment for an example. Here, each person's name is a data element, and your job is to copy each name to its own string:

```
var DataString = 'Bob Smith, Mary Jones, Tom Roberts, Sue Baker‘
```

split() method

It can easily be accomplished by using the split() method of the string object. The split() method creates a new array and then copies portions of the string, called a substring, into its array elements. You must tell the split() method what string (delimiter) is used to separate substrings in the string. You do this by passing the string as an argument to the split() method. In this example, the comma is the string that separates substrings.

split() method Example

Syntax:

```
var NewStringArray = StringName.split('delimiter string')
```

Example:

```
var DataString = 'Bob Smith,Mary Jones,Tom Roberts,Sue Baker'
var NewArray = DataString.split(',')
document.write(DataString); document.write('')
for (i=0; i<4; i++)
{
 document.write(NewArray[i])
document.write('<br>')
}
```

Copying a sub string

Consider the following email address

```
EmailAddress = 'bsmith@xyz.com '
```

Now if you want to extract only xyz.com from above email address so that can be stored as website address.

To complete it you can use substring() and substr() methods.

substring() method

Syntax:

```
var NewSubstring = StringName.substring (StartingPosition, EndPosition)
```

starting position specifies the first character that is returned by the sub- string() method.

The **end position** specifies the position of the character that comes after the last character that you want to include in the substring.

Example for substring method

```
var EmailAddress = 'sameer@myweb.com'
var NewSubstring = EmailAddress.substring(7,16)
var WebSite = 'www.' + NewSubstring
console.log(WebSite)  // www.myweb.com
```

---

substr() method

The substr() method returns a substring. You must tell it the starting position of the fi rst character that you want to include in the substring and how many characters you want copied into the substring from the starting position. Both positions are passed as arguments to the substr() method. Here's how you write the substr() method:

Syntax:

```
Var NewSubstring = StringName.substr (StartingPosition, NumberOfCharactersToCopy)
```

Example:

```
var EmailAddress = 'sameer@myweb.com'
var NewSubstring = EmailAddress.substr(7,9)
var WebSite = 'www.' + NewSubstring
console.log(WebSite)  // www.myweb.com
```

Converting Numbers and Strings

If you need to convert string values to number values, you can do so by converting a number within a string into a numeric value that can be used in a calculation. You do this by using the **parseInt()** method and **parseFloat()** method of the string object.

You cannot use this number in a calculation because '100' is a string and not a numeric value that is, the browser treats this as text and not a number. You must convert this string to a numeric value before you can use the 100 in a calculation. The following statement is used for this conversion:

```
var StrCount = '100'

var NumCount = parseInt(StrCount)
```

The parseFloat() method is used similarly to the parseInt() method, except the parseFloat() method is used with any number that has a decimal value. (Think of a decimal number whenever you see the word fl oat.) Here's how to use the parseFloat() method:

```
var StrPrice = '10.95'

var NumPrice = parseFloat(StrCount)
```

Numbers to Strings

you need to convert a numeric value to a string You do this by calling the toString() method of the number object. The toString() method can be used to convert both integers and decimal values (fl oats). Here's how to convert a number value to a string:

```
var NumCount = 100

var StrCount = NumCount.toString()
```

Changing the Case of the String

Alternatively, you can use the concatenation operator (+) to combine a string and a number. The concatenation operator automatically calls the toString() method on numeric values to convert them to a string. This is illustrated here:

```
var x = 500

var y = 'abc'

var z = x + y
```

Variable z now has the value `'500abc'` and has been converted to a string.

Changing the Case of the String

Changing the case of the string is done by using the toUpperCase() method and toLowerCase() method of the string object. The functions return a new string that's either all uppercase or all lowercase. The original string is unchanged.

```
var C1 = 'SaMeer'

console.log(C1.toLowerCase())

Output:
        sameer
```

```
var C1 = 'SaMeer'
var C2 = 'sameer'
if (C1.toUpperCase() == C2.toUpperCase())
        {
        console.log('converted to upper case')
        }
Output:
        converted to upper case
```

Strings and Unicode

You probably already know that a computer understands only numbers and not characters. You might not know that when you enter a character, such as the letter w, the character is automatically converted to a number called a Unicode number that your computer can understand.

- You can determine the Unicode number by using the charCodeAt() method.
- You can determine the character that is associated with a Unicode number by using fromCharCode() method.
- Both are string object methods.

The **charCodeAt()** method takes an integer as an argument that represents the index of the character in which you're interested. If you don't pass an argument, it defaults to index 0.

Syntax

```
var UnicodeNum = StringName.charCodeAt()
```

Example

```
var S1 = 'w'
var S2 = 'polytechnic'
var Unicode1 = S1.charCodeAt()
var Unicode2 = S2.charCodeAt(6)

console.log(Unicode1) // output : 97
console.log(Unicode2) // output : 99
```

If you need to know the character, number, or symbol that is assigned to a Unicode number, use the **fromCharCode()** method.

The fromCharCode() method requires one argument, which is the Unicode number.

Here's how to use the fromCharCode() method to return the letter w.

```
var L = String.fromCharCode(119)

console.log(L)  // output : w
```

Since we don't have any string object to call fromCharCode() method, so that we called it by class name `string`.