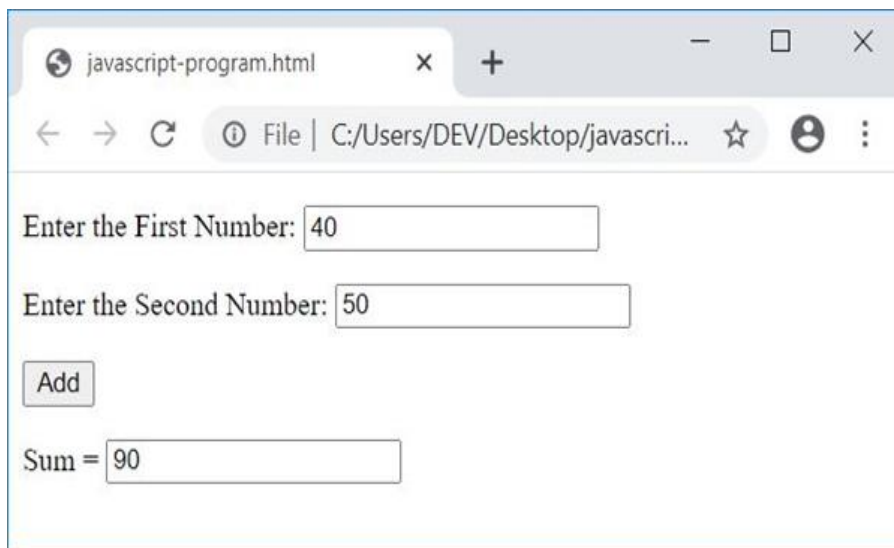


## Practical -9

### Operators Function and Array in JavaScript

1. Write a javascript to test the following arithmetic[ +, -, \*, /, %] and comparison operators [==, !=, >, <, >=, <=]
2. Write a function to reverse a given number.
3. Write a javascript to check global and local scope of a variable.
4. Write a recursive function to calculate factorial of a number.
5. Write a JavaScript for passing a variable as a parameter to a function call, and another that passes the return value of a function directly to the parameters of another function [Hint perform multiplication of two numbers]
6. Write a javascript to add two numbers using form and textbox as shown below : [hint use **a=document.getElementById(id of textbox).value** to refer the value inputted by user. For display of result use **document.getElementById(id of textbox).value = s**]



The screenshot shows a web browser window with the title 'javascript-program.html'. The address bar shows the file path 'C:/Users/DEV/Desktop/javascript...'. The page content includes two input fields for numbers, an 'Add' button, and a 'Sum =' label followed by a text box displaying the result '90'.

Enter the First Number:

Enter the Second Number:

Sum =

7. Write a javascript to print even numbers in a given array
8. Write a Javascript to find the largest and smallest number in an array.

### JavaScript Functions

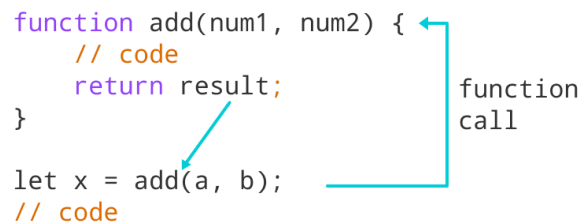
In JavaScript, a function allows you to define a block of code, give it a name and then execute it as many times as you want. A JavaScript function can be defined using **function** keyword.

#### Syntax:

```
//defining a function
function <function-name>()
{
    // code to be executed
};
```

```
//calling a function
<function-name>();
```

Functions *always* return a value. In JavaScript, if no return value is specified, the function will return undefined.



```
function add(num1, num2) {
    // code
    return result;
}

let x = add(a, b);
// code
```

### Function Parameters

A function can have one or more parameters, which will be supplied by the calling code and can be used inside a function. JavaScript is a dynamic type scripting language, so a function parameter can have value of any data type.

**Note :** 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.

#### Example: Function Parameters

```
function ShowMessage(firstName,lastName) {
    alert("Hello " + firstName + " " + lastName);
}
```

```
ShowMessage("Steve","Jobs");
ShowMessage("Bill","Gates");
ShowMessage(100,200);
```

### Default Parameters

If a function is called with **missing arguments** (less than declared), the missing values are set to undefined. Sometimes this is acceptable, but sometimes it is better to assign a default value to the parameter:

#### Example

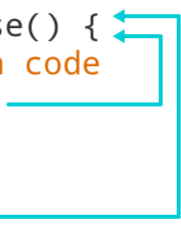
```
function myFunction(x, y = 10) {
    return x + y;
}

myFunction(5);
```

## Recursive function

```
function recurse() {
  // function code
  recurse();
}

recurse();
```



## Example

```
function add(number) {
  if (number <= 0) {
    return 0;
  } else {
    return number + add(number - 1);
  }
}

add(3) => 3 + add(2)
        3 + 2 + add(1)
        3 + 2 + 1 + add(0)
        3 + 2 + 1 + 0 = 6
```

## Local and Global Variables

**Scope** in JavaScript defines accessibility of variables, objects and functions.

Variables declared outside of any function become global variables. Global variables can be accessed and modified from any function.

Variables declared inside any function with **var** keyword are called local variables. Local variables cannot be accessed or modified outside the function declaration.

Variables declared inside a function without **var** keyword also become global variables.

## Array

An *array* can be constructed with square brackets or `Array(____)` or `new Array(____)` or `Array.of(____)` or `Array.from(____)`. You will use square brackets 99% of the time or more. It has a `length` property as well as non-negative integer properties.

```
const a = []
```

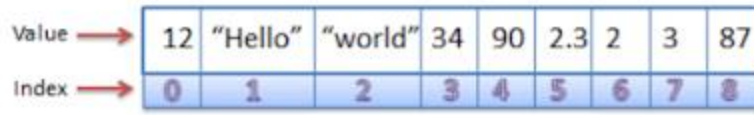
```
let b = [1, 5, 25]
```

```
let c = new Array() // Same as []
```

```
c = new Array(4, 5, 6) // Same as [4, 5, 6]
```

```
c = new Array(10) // 10 empty cells
```

```
const [d, e, f] = b // d===1, e===5, f===25 (destructuring)
```



The diagram illustrates a JavaScript Array as a horizontal table with two rows. The top row is labeled 'Value' with a red arrow pointing to it, and contains the elements: 12, "Hello", "world", 34, 90, 2.3, 2, 3, and 87. The bottom row is labeled 'Index' with a red arrow pointing to it, and contains the corresponding indices: 0, 1, 2, 3, 4, 5, 6, 7, and 8. Below the table, the text 'JavaScript Array' is centered.

Value →	12	"Hello"	"world"	34	90	2.3	2	3	87
Index →	0	1	2	3	4	5	6	7	8

JavaScript Array

A JavaScript array has the following characteristics:

1. First, an array can hold values of different types. For example, you can have an array that stores the number and string, and boolean values.
2. Second, the length of an array is dynamically sized and auto-growing.

### Accessing the Elements of an Array

Array elements can be accessed by their index using the square bracket notation. An index is a number that represents an element's position in an array.

Array indexes are zero-based. This means that the first item of an array is stored at index 0, not 1, the second item is stored at index 1, and so on. Array indexes start at 0 and go up to the number of elements minus 1. So, array of five elements would have indexes from 0 to 4.

The following example will show you how to get individual array element by their index.

#### Example

```
var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
document.write(fruits[0]); // Prints: Apple
document.write(fruits[1]); // Prints: Banana
document.write(fruits[2]); // Prints: Mango
```

### Getting the Length of an Array

The length property returns the length of an array, which is the total number of elements contained in the array. Array length is always greater than the index of any of its element.

#### Example

```
var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
document.write(fruits.length); // Prints: 5
```

### Looping Through Array Elements

You can use for loop to access each element of an array in sequential order, like this:

#### Example

```
var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
// Iterates over array elements
for(var i = 0; i < fruits.length; i++) {
    document.write(fruits[i] + "<br>"); // Print array element
}
```

## Multidimensional Array

## 1-D array

arr = [1, 2, 3, 4, 5]

(0)

1	2	3	4	5
---	---	---	---	---

arr[0], arr[1], arr[2]

## 2-D array

```
arr = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
```

(0, 0)

1	2	3
4	5	6
7	8	9

```
arr[0][0] => 1
arr[1][2] => 6
arr[2][0] => 7
```

## 3-D array

```
arr = [
    [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ],
    [ [10, 11, 12], [13, 14, 15], [16, 17, 18] ],
    [ [19, 20, 21], [22, 23, 24], [25, 26, 27] ],
]
```

(0, 0, 0)

1	2	3		
4	5	6		
7	8	9		

```
arr[0][0][0] => 1
arr[1][0][0] => 2
arr[0][1][0] => 4
arr[0][0][1] => 10
```