Call by Value Vs Call by Reference in JavaScript

Last Updated : 25 Apr, 2023

Call by Value: In this method, values of actual parameters are copied to the function's formal parameters, and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in the actual parameters of the caller.

Suppose there is a variable named **"a"**. Now, we store a primitive value(boolean, integer, float, etc) in the variable **"a"**. Let us store an integer value in **"a"**, Let a=5. Now the variable **"a"** stores 5 and has an address location where that primitive value sits in memory.

Now, suppose we copy the value of **"a"** in **"b"** by assignment (**a=b**). Now, **"b"** points to a new location in memory, containing the same data as variable **"a"**.

Thus, a=b=5 but both point to separate locations in memory.

This approach is called **call by value** where 2 variables become the same by copying the value but in 2 separate spots in the memory.

Features of call by value:

- Function arguments are always passed by value.
- It copies the value of a variable passed in a function to a local variable.
- Both these variables occupy separate locations in memory. Thus, if changes are made in a particular variable it does not affect the other one.

Example: This example shows the use of the above-explained approach.

JavaScript

```
// By value (primitives)
let a = 5;
let b;
b = a;
a = 3;
console.log(a);
console.log(b);
```

Output: **"b"** was just a copy of **"a"**. It has its own space in memory. When we change **"a"** it does not have any impact on the value of **"b"**.

```
3
5
```

Call by reference: Let's say, we have an object stored in the variable **"a"**. The variable stores the location or the address where the object lives. Now we set **b=a**. Now that new variable **"b"** instead of pointing to a new location in the memory, points to the same location where **"a"** does. No new object is created, and no copy is created. Both variables point to the same object. This is like having 2 names.

This is **called by reference**. It behaves quite differently from by value. All objects interact by reference.

Features of By reference:

- In JavaScript, all objects interact by reference.
- If an object is stored in a variable and that variable is made equal to another variable then both of them occupy the same location in memory.
- Changes in one object variable affect the other object variable.

Example: Over here, when we set **d=c**, **"d"** points to the same location in memory where **"c"** does. At first, we have a name-value pair stored in **"c"**. Now when we change a property using **"c"**, it changes the property in **"d"** also because both point to the same object. Changes in one it affects.

Javascript

```
// By reference (all objects (including functions))
let c = { greeting: 'Welcome' };
let d;
d = c;

// Mutating the value of c
c.greeting = 'Welcome to geeksforgeeks';
console.log(c);
console.log(d);
```

Output:

```
{greeting: 'Welcome to geeksforgeeks'}
{greeting: 'Welcome to geeksforgeeks'}by
```

Difference between call by value and call by reference:

Call by value	Call by reference
The original variable is not modified on changes in other variables.	The original variable gets modified on changes in other variables.
Actual and copied variables will be created in different memory locations.	Actual and copied variables are created in the same memory location.
On passing variables in a function, any changes made in the passed variable will not affect the original one.	On passing variables in a function, any changes made in the passed parameter will update the original variable's reference too.