# **Primitive vs Reference Data Types in JavaScript**

**Primitive Data Type**

Primitive data Type are static values

Let Num\_1=10; ==> static value( Means we can not add , update …)

Region for this JavaScript knows how much space required for storing this value in memory.

Primitive values directly stored in stack memory but reference value (Array, Object, Function ) stored in heap memory.

When you declare a primitive data type in JavaScript, it is stored on a stack. A stack is a simple data structure that the computer uses to store and retrieve data quickly.

// Primitive Data type vs Reference Data type

let Num\_1=50;

let Num\_2= Num\_1;

console.log("Value of Num-1 is " ,Num\_1);

console.log("Value of Num-2 is " ,Num\_2);

Num\_1=30;

console.log("After update the value of Num-1")

console.log("Value of Num-1 is " ,Num\_1);

console.log("Value of Num-2 is " ,Num\_2);

**Stack Memory**

**Both Contain separate space in stack memory that’s why no effect on Num\_2 .if I update the value of Num\_1.**

Num\_2=50;

Num\_1=50;

Suppose we declare a variable, Num\_1, and give it a value of 50. We go on to create another variable, Num\_2, and assign it Num\_1 in Num\_2 . So both variables have the same value.

What happens on the stack is that, the computer creates room for Num\_1 and stores its assigned value on the stack. Again creates room for Num\_2 and stores the value of Num\_1 on the stack. It does not matter that both variables are assigned the same value.

What if during the coding process, we decided to update the value of Num\_1  to say, 30? Does it mean  Num\_2 will change too? The answer is no.

Since  Num\_1  and  Num\_2  were stored differently on the stack, because of that if I updating one of them will not affect the other.

**Reference Data Type**

Primitive data Type are dynamic values

Let arr1=[4, 5, 6, 8]; ==> Dynamic value( Means we can add , update , delete…)

Region for this JavaScript can not determine how much space required for storing this value in memory.

//Reference Data type

let arr1=[2,4,6,8];

let arr2=arr1;

console.log("Value of arr1 is " ,arr1);

console.log("Value of arr2 is ",arr2);

arr1.push(9,10);

console.log("After Adding the value in arr1");

console.log("Value of arr1 is " ,arr1);

console.log("Value of arr2 is " ,arr2);

**Stack Memory Heep Memory**

Array value store in heap

[2,4,6,8]

[2,4,6,8,9,10]

Address- 10010

Pointer of arr2(add-10010)

Pointer of arr1

Update (add-10010)

Arr.push(9, 10)

Num\_1=50;

Suppose we create an array I.e arr1 and assign it a value that is a reference data type, the computer does not directly store that value in that variable (as is the case with primitive types) it store the value of variable in heap and its pointer is stored on a stack. And The pointer contain the address of arr1 values .

Now, we could create arr2 variable , and assign an arr1 in arr2 . Does that mean arr2 will be created on the heap? The answer is no.

Since the arr1 values already exists on the heap,  arr1 and arr2  will both point to the arr1 values address. If I update arr1 and try to print arr2 in that case arr2 contain same value which are update in arr1. This is because they are pointing to the same arr1 on the heap – and updating one variable of course affects the other.