**BY VALUE VS BY REFERENCE:**

In both cases, we are talking about variables.

**Important: Don’t forget “=” equals sign creates a new spot in the memory and if its primitive value then it will copy the value and assign it into the new spot – which it created.**

By Value:

Let’s say I have a primitive value *(number, Boolean or a string). by value (primitives).*

Variable A

Primitive Value

------------------------------------

Memory location – 0 x 001

Now variable “a”, which has a primitive value, has an address location. If we set up a new variable and make it equal to variable “a”.

b = a

If it’s a primitive value then B the new variable point to a new address, a new location in the memory and a copy of the primitive value is placed into that spot in memory.

Var “B”

Var “A”

------------------------------------------

Copies value of A then assign

It into a new memory location

ML – 0 \* 001 = COPY PRIMITIVE VALUE = ML – 0 \* 002

This approach is called “By Value”.

Example:

var a =3; // remember 3 is a primitive type a number

var b;

b = a ;

(var b will copy the value from a and it will be stored in different memory location).

Which means after var “a” value is copied then it shouldn’t affect the value which variable “b” holds.

a = 2;

console.log(a);

console.log(b);

var “a “will return 2 and var “b” will return 3.

By Reference:

*This is about (all objects – including functions)*.

**Example:**

var c = {

greeting: 'hello'

}

var d;

d = c;

Memory Location

Alias Alias

var d

var c

c.greeting = 'Hello Mr Ali Khan'; */\* Mutate meaning change \*/*

console.log(c); */\* it does not matter whichever variable either (c or d) you call they both are just an alias to a memory location \*/*

console.log(d);

*/\* They both are pointing towards the same memory location, when we are using objects - this is called by reference. \*/*

*in objects, if you change one that means you change all of them.*

BY REFERENCE (even as parameter)

function changeGreetings(obj) {

obj.greeting = "Hola";

}

changeGreetings(d);

console.log(c);

console.log(d);

As mentioned above, even using a parameter in a function, we managed to mutate the object and via console you can check once you mutate the object the memory reference will be updated for all of them.

Equals operator sets up new memory space (new memory address)

c = {

greeting: "Howdy"

}

console.log(c);

console.log(d);

This is a special case where by reference doesn’t apply because the equals to = saw that these aren’t a pre-existing location in memory. This is a brand new object created on the fly by the “object literal syntax – {}”.

# Conclusion:

All primitive values are works by value &

All objects (including functions) are by reference.