CH3: Is JavaScript pass by reference?

```
*REFER TO PHYSICAL NOTES FOR TABLES*
    -say we have:
        let n1 = 31
        let n2 = n1
        n1 = 32
        clg(n2) = ??
    My initial thought would be that it would be 31. I think this because my
intutition says it is pass by value;
    ANS= 31

        let f1 = {num: 3, den: 4}
        let f2 = f1
        f1 = {num:1, den: 2}
        clg(f2)
    My initial thought would be that console logs {nums:1, den:2} because it is a
reference to an object
    ANS= is is 3 / 4 BECAUSE when we evaluate a expression in JS, we do the right
hand side first and then we assign it
    to the left. What happens here is that we are taking the right hand side of
{num:1, den:2} and creating a whole new object and then assigning it to f1. What
happens under the hood is that since we are technically creating a new object, we
need a brand new memory addy on the heap that has to get created and then takes
that new addy and assigns it to f1. So nothing with the og 3/4 gets change, we
just have a brand new object being created

        let f1 = {nums:3, dec:4}
        let f2 = f1
        f1.num = 1
        f1.den = 2
        clg(f2)
    I think it will print {1/2}
    Ans= what I said ^. This happens because f1 and f2 under the hood point to the
same address and so when we
    update that object from f1, f2 will see those same changes since they are
pointing to the same address

        let f1 = {num: 3, dec: 4}
        let f2 = f1
        f1.toDecimal(function(){
            return this.num / this.den
        })
        clg(f2.toDecimal)
        will this work?
        Initial thought no since this does not exist in our object
        ANS= YES IT WILL work. Because it creates a new function that is stored on
the heap and that function has
        a pointer from the num object pointing to it as a reference so it would
```

```
work since f1 and f2 point to the same thing.


Objects are stord as MEMORY. This can be important when we are trying to copy
another object and we change values
around. Variable declaration without let/ const are mean it is global. OBV, global
is bad (for the most part).
```

CH4: Lexical block scope: - variables are declared with let/ const - variables are defined under the most recent scope operator Higher order functions: - Functions that are considered either one that: - take a function as an input - returns as an output a function - both * Closure: is preservation of access to variables that are otherwise out of scope. That an inner funciton or block of code has a dependancy. It has a pointer to the variable address. Only the variables that are used in the inner functions are saved, anything that is unused will, be put out for garbage collection.