Deep copy vs Shallow Copy

Arrow Function

- Arrow function allows us to write shorter function syntax.

E.g. Suppose we have to write a function to multiply two numbers.
We might usually write as:

```
var ans = function(x,y){
    return x * y;
}
```

But with the help of the arrow function, we can rewrite in an easy way. We can omit the function in the above syntax and write arguments directly.

```
var ans = (x,y) => {
    return x * y;
}
```


It gets even shorter!
If the function has only one statement, and the statement returns a value, you can remove the brackets *and* the return keyword like:

```
var ans = (x,y) => {
    return x * y;
}
var ans = (x,y) => x*y;
console.log(ans(2,3));
```

The output will be:

```
6
```

Deep Copy

- In Deep copy, The values that are copied in the new variable are copied and disconnected from the original variable, which means if you try to change the new variable, the original variable shouldn't have any changes.
- For a primitive value, you use a simple assignment:

For e.g.

```
let num = 5;
let copiedValue = num;
```

And when you change the value of the copiedValue variable, the original value remains the same.

```
copiedValue = 10;
console.log(num); //5
```

How to make a Deep copy of arrays or objects?

By Using Spread Operator

The spread operator (...) allows us to copy all or part of an existing array or object into another array or object.

E.g.

```
var susan = {
    name : 'Susan',
    age: 30
};

console.log("Before using spread operator");
console.log(susan);

var copiedSusan = {
    ...susan
}

copiedSusan.name = "Raj";
console.log("After using spread operator");
console.log(copiedSusan);
console.log(susan);
```

The Output will be:

```
Before using spread operator
{ name: 'Susan', age: 30 }
After using spread operator
{ name: 'Raj', age: 30 }
{ name: 'Susan', age: 30 }
```

In this example, all values in the copiedSusan object are disconnected from the original susan object. So "susan" and "copiedSusan" both are different objects.

Note: Spread operator can be used in many cases, like when we want to expand, copy,concat with math object.

Also, we have another method to create a deep copy of an object:
Using Assign method:

Syntax:

```
Object.assign(target, ...sources)
```

For e.g.

```
var virat = Object.assign({}, susan);
console.log(virat);
virat.name = "virat";
virat.age = 45;
console.log(virat);
```

The output will be:

```
{ name: 'Susan', age: 30 }
{ name: 'virat', age: 45 }
```

Shallow Copy:

- In shallow copy, when we copy the original object into the new object, the new object has the copy of the original object's memory address, which means both objects point to the same memory address.

For e.g.

```
var person1 = {Name : 'John', Age : 25};
var person2 = person1;
```

If you assign one object to another like person2 = person1, then the value of ‘person1’ gets assigned to ‘person2’ and therefore, they both will point to the same location.

```
console.log(person1 == person2); //return true
console.log(person1 === person2); //return true
```

Now if you changed the person2 object, changes would be reflect bcak to the original ‘person1’ object because they point to the same memory address.

```
person2.State = "Delhi";
console.log(person2);
console.log(person1);
```

The output will be:

```
{ Name: 'John', Age: 25, State: 'Delhi' }
{ Name: 'John', Age: 25, State: 'Delhi' }
```