**Task 2**

**1. Difference between Copy by value and Copy by reference in JS**

|  |  |
| --- | --- |
| Copy by Value | Copy by Reference |
| It applies to Primitive data types such as Number, Bool, String | It applies to composite data types such as Array, Objects and Functions |
| If a primitive data type is assigned to another variable, only the copy of the value is assigned i.e the original value is copied and new value is assigned | If a composite data type is assigned to another variable, the reference of it is assigned to the variable i.e. the location at which the value is stored is assigned |
| If a change is made in the original value, it will not affect the newly assigned variable | If a change is made in the original value, it will affect both the original and newly assigned variable |
| When the equality operators, == and ===, are used on value-type variables, they check the values. If the variables contain the same value, the comparison will result in true. | When the equality operators, == and ===, are used on reference-type variables, they check the reference. If the variables contain a reference to the same item, the comparison will result in true. |

**2. Copy by value a composite data type**

**1. Using Spread**

Spread operator allows an iterable to expand in places where 0+ arguments are expected. It is mostly used in the variable array where there are more than 1 values are expected. It allows us the privilege to obtain a list of parameters from an array. Using spread will clone your object. Note this will be a shallow copy.

Example:

var a = [1, 2, 3];

var c = […a];

console.log(a, c); => (3)[1, 2, 3] (3)[1, 2, 3]

c[2] = 4;

console.log(a, c); => (3)[1, 2, 3] (3)[1, 2,4]

In the above example when copied variable value is changed but original variable value remains same.

**2. Using Object.assign()**

The Object.assign() method copies all enumerable own properties from one or more source objects to a target object. It returns the target object. Note this will be a shallow copy.

Example:

var a = [1, 2, 3];

var c = Object.assign([], a);

console.log(a, c); => (3)[1, 2, 3] (3)[1, 2, 3]

c[2] = 4;

console.log(a, c); => (3)[1, 2, 3] (3)[1, 2,4]

Note the empty [] as the first argument, this will ensure you don't mutate the original object

**3. Using JSON.parse() and JSON.stringify()**

The JSON object, available in all modern browsers, has two useful methods to deal with JSON-formatted content: parse and stringify. JSON.parse() takes a JSON string and transforms it into a JavaScript object. JSON.stringify() takes a JavaScript object and transforms it into a JSON string.Using JSON.parse() and JSON.stringify() for copy performs deep copy .

Example:

var a = [1, 2, 3];

var c = JSON.parse(JSON.stringify(a);

console.log(a, c); => (3)[1, 2, 3] (3)[1, 2, 3]

c[2] = 4;

console.log(a, c); => (3)[1, 2, 3] (3)[1, 2,4]

The deep copy is a true copy for nested objects. Shallow copy copies only reference in case of nested objects.