2. Give a write-up on the Difference between copy by value and copy by reference.

**Copy by value:**

Primitive datatypes are passed, or copied, by value and are immutable, meaning that the existing value cannot be altered the way an array or an object can.

var a = 10;

var b= 20;

var c = a;

var d = b;

console.log(a) //10

console.log(b) //20

console.log(c) //10

console.log(d) //20

c= 100;

console.log(a,c) // 10 100

here the value of **c** has been changed but the value of **a** remains the same,thus it copies only the value.

**Copy by Reference:**

Objects, on the other hand, are passed**by reference** and point to a location in memory for the value, not the value itself.

var obj ={name:'banu'}

var y = obj;

console.log(y.name) //banu

y.name = "priya"

console.log(obj); //priya

here when the copied variable value is changed, it has affected the original variable itself,which shows objects copy the value with their addressand hence its mutable.

1. Load the rest countries' data using your HTML and script.js file and run a for loop on the data and print all the country names in the console.

**HTML FILE:**

<!DOCTYPE html>

<html lang="en">

<head>

<title>api call</title>

</head>

<body>

<h1>country names in the console</h1>

<div id="root"></div>

<script src="request.js">

</script>

</body>

</html>

**JAVASCRIPT FILE:**

var request = new XMLHttpRequest()

var url\_string = "https://restcountries.eu/rest/v2/all";

request.open('GET',url\_string , true)

request.send();

request.onload = function() {

var data = JSON.parse(request.response)

for(i=0;i<data.length;i++)

{

console.log(data[i].name);

}

}

3. How to copy by value a composite datatype (array+objects).

There are 3 ways to copy by value for composite data types.

1. Using the spread ( ... ) operator.
2. Using the Object.assign() method.
3. Using the JSON.stringify() and JSON.parse() methods.

**1.using spread operator:**

var a = [1,2,3];

var c=[...a];

console.log(a,c) //[ 1, 2, 3 ] [ 1, 2, 3 ]

c[0] = 1000;

console.log(a,c) //[ 1, 2, 3 ] [ 1000, 2, 3 ]

here the spread operator helps in copying the value without its reference hence making it as an **shallow copy** of the original array. Note: In nested objects, shallow copy copies the reference and hence it does not works well with nested object.

**2.using Object.assign():**

var a = [1,2,3];

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

console.log(a,c) //[ 1, 2, 3 ] [ 1, 2, 3 ]

c[2] = 999;

console.log(a,c) //[ 1, 2, 3 ] [ 1, 2, 999 ]

this method copies all enumerable own properties from source to targe objects.this will also be a **shallow copy** of the source. Note: In nested objects, shallow copy copies the reference and hence it does not works well with nested object.

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

var a = [1,2,3];

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

console.log(a,c) //[ 1, 2, 3 ] [ 1, 2, 3 ]

c[1] = 54535 ;

console.log(a,c) //[ 1, 2, 3 ] [ 1, 54535, 3 ]

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 .

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