Task 2:

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 name in the console.

The above task can be done using the following index.html and script.js files,

index.html file

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script src="script.js"></script>
</body>
</html>
```

script.js file

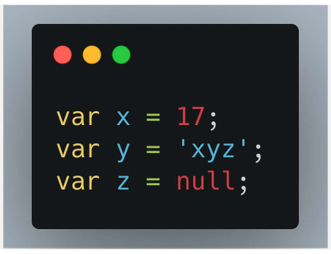
```
var request=new XMLHttpRequest();
request.open('GET', 'https://restcountries.eu/rest/v2/all', true)
request.send();
request.onload = function(){
    var countrydata = JSON.parse(this.response);

    for(var i=0;i<250;i++){
        console.log(countrydata[i].name);}
}
```

2. Write a write up on Difference between copy by value and copy by reference.

**Copy by value**

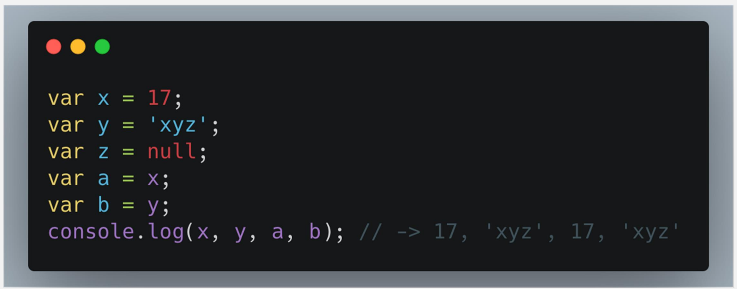
In a primitive data-type when a variable is assigned a value we can imagine that a box is created in the memory. This box has a sticker attached to it i.e. the variable name. Inside the box the value assigned to the variable is stored.



```
var x = 17;
var y = 'xyz';
var z = null;
```

Figure 1: variable assignment

In Figure 1, 'x' contains value 17, 'y' contains 'xyz'.



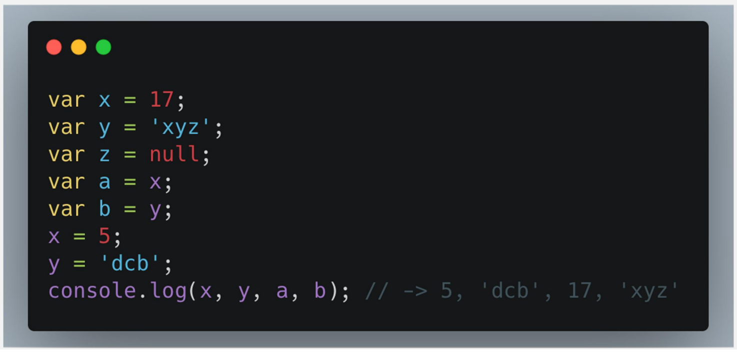
```
var x = 17;
var y = 'xyz';
var z = null;
var a = x;
var b = y;
console.log(x, y, a, b); // -> 17, 'xyz', 17, 'xyz'
```

Figure 2: **Variables are copied**

In the Figure 2; the **values** in the boxes 'x' and 'y' are copied into the variables 'a' and 'b'. At this point of time both 'x' and 'a' contains the value 17. Both 'y' and 'b' contains the value 'xyz'. However, an important thing to understand here is that even though 'x' and 'a' as well as 'y' and 'b' contains the same value they are not connected to each other. **It is so because the values are directly copied into the new variables.**

We have seen that, in a primitive data-type, when a variable is assigned a value, a box is created in the memory. This box has a sticker attached to it i.e. the variable name. Inside the box the value assigned to the variable is stored. Hence here two new boxes (memory boxes for new variables) are created.

Changes taking place in one does not affect the other.

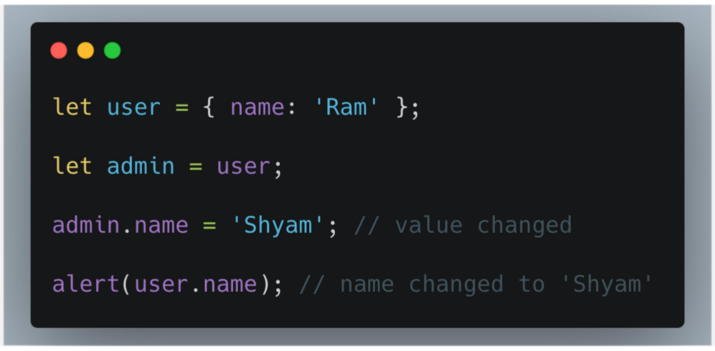


```
var x = 17;
var y = 'xyz';
var z = null;
var a = x;
var b = y;
x = 5;
y = 'dcb';
console.log(x, y, a, b); // -> 5, 'dcb', 17, 'xyz'
```

Figure 3: **Variables are independent of each other**

## Copy by reference

**In case of a non-primitive data-type the values are not directly copied**. When a non-primitive data-type is assigned a value, a box is created with a sticker of the name of the data-type. However, the values it is assigned is not stored directly in the box. The language itself assigns a different memory location to store the data. The address of this memory location is stored in the box created.


```
let user = { name: 'Ram' };

let admin = user;

admin.name = 'Shyam'; // value changed

alert(user.name); // name changed to 'Shyam'
```

Figure 4: **Copy by reference**

In the Figure 3, when the value of admin is changed it automatically changes the value of user as well.

This happens because both 'user' and 'admin' are storing the address of the memory location. And when one changes the values in the allocated memory it is reflected in the other as well.

**We can further elaborate it we can say that; copy by reference is like having two keys of the same room shared between 'admin' and 'user'. If one of them alters the arrangement of the room, the other would experience it ads well.**

The above seen are the key differences between copy by value and copy by reference.

## 3. How to copy by value a composite data type (array + objects).

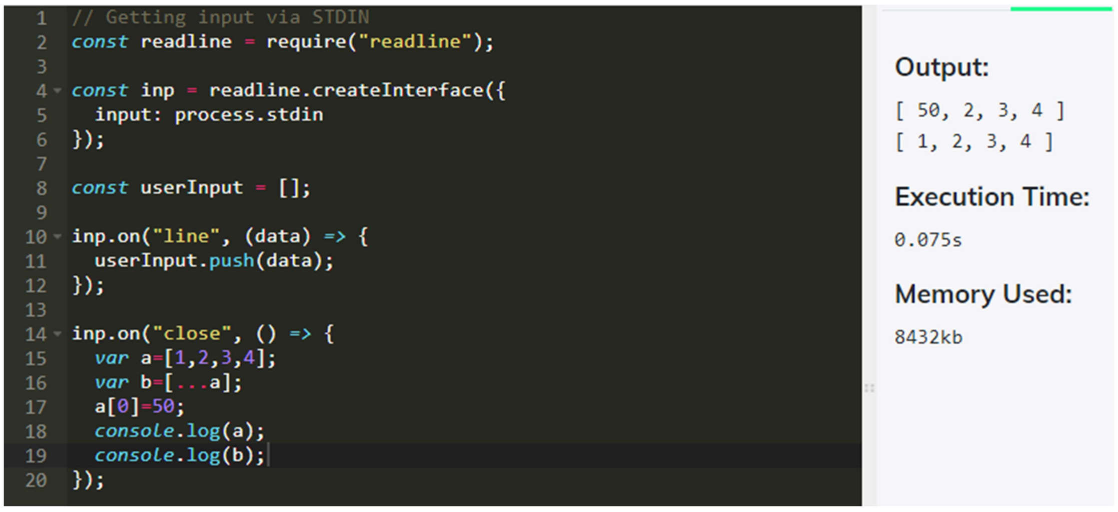
Spread operator can be used to copy by value a composite data type.

Arrays and objects are object types which come under composite data types. As we know, for primitive data types, variable holds data. In case of composite data type, the variable holds the memory location of that particular value.

The following examples show how values get assigned to variables for primitive ad composite data types.

var a= 10; // here 'a' holds the value 10.

var b= [10,20] // 'b' holds some address like 8023 etc.. not 10 and 20.



```
// Getting input via STDIN
const readline = require("readline");

const inp = readline.createInterface({
  input: process.stdin
});

const userInput = [];

inp.on("line", (data) => {
  userInput.push(data);
});

inp.on("close", () => {
  var a=[1,2,3,4];
  var b=[...a];
  a[0]=50;
  console.log(a);
  console.log(b);
});
```

**Output:**

```
[ 50, 2, 3, 4 ]
[ 1, 2, 3, 4 ]
```

**Execution Time:**

```
0.075s
```

**Memory Used:**

```
8432kb
```

Figure 1

Therefore we can’t clone data in composite data types. To do that spread operator is used, that is, three dots (…). This operator spreads the elements of a particular array or object so that its values can be assigned to some other variable as shown in figure 1.

*-------------------------------------*