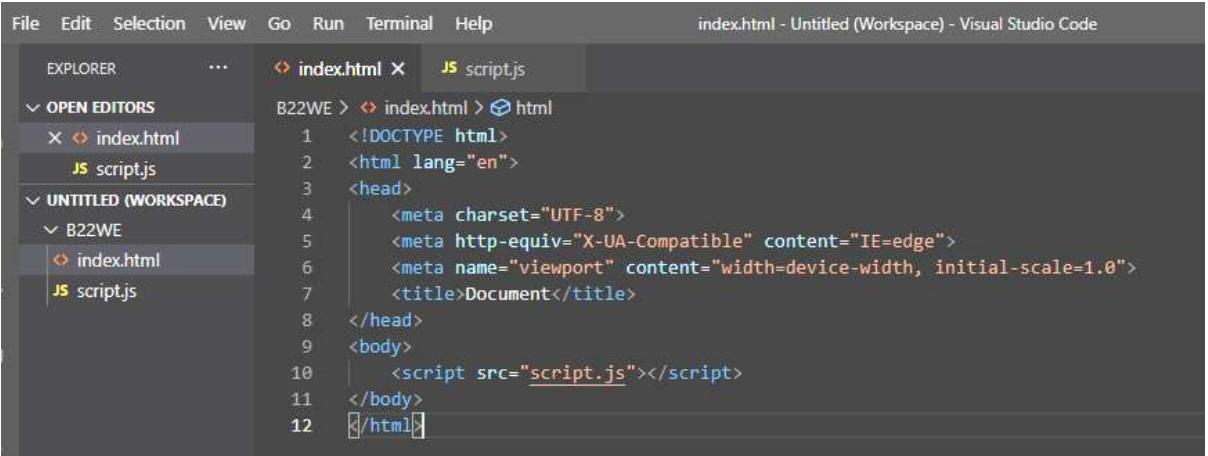


TASK 2- 20th March

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.

a) To load all the data of rest countries

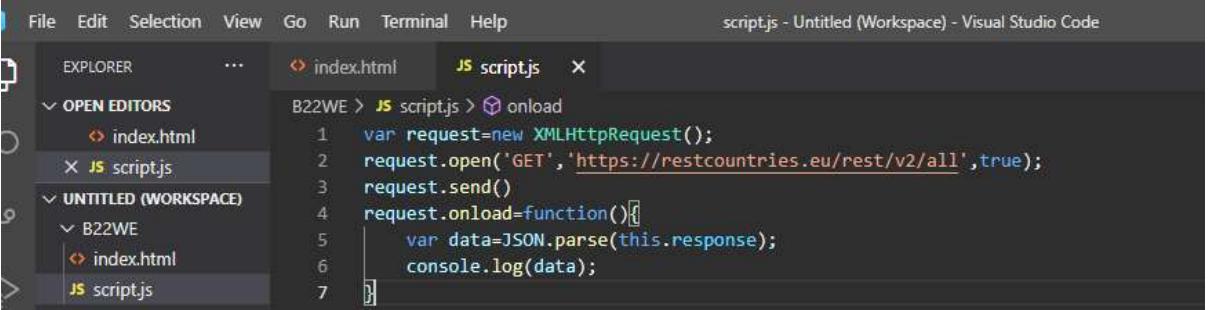
Index.html



```
File Edit Selection View Go Run Terminal Help
index.html - Untitled (Workspace) - Visual Studio Code

EXPLORER ... index.html JS script.js
OPEN EDITORS B22WE > index.html > html
X index.html JS script.js
UNTITLED (WORKSPACE) B22WE
index.html JS script.js
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>Document</title>
8 </head>
9 <body>
10    <script src="script.js"></script>
11 </body>
12 </html>
```

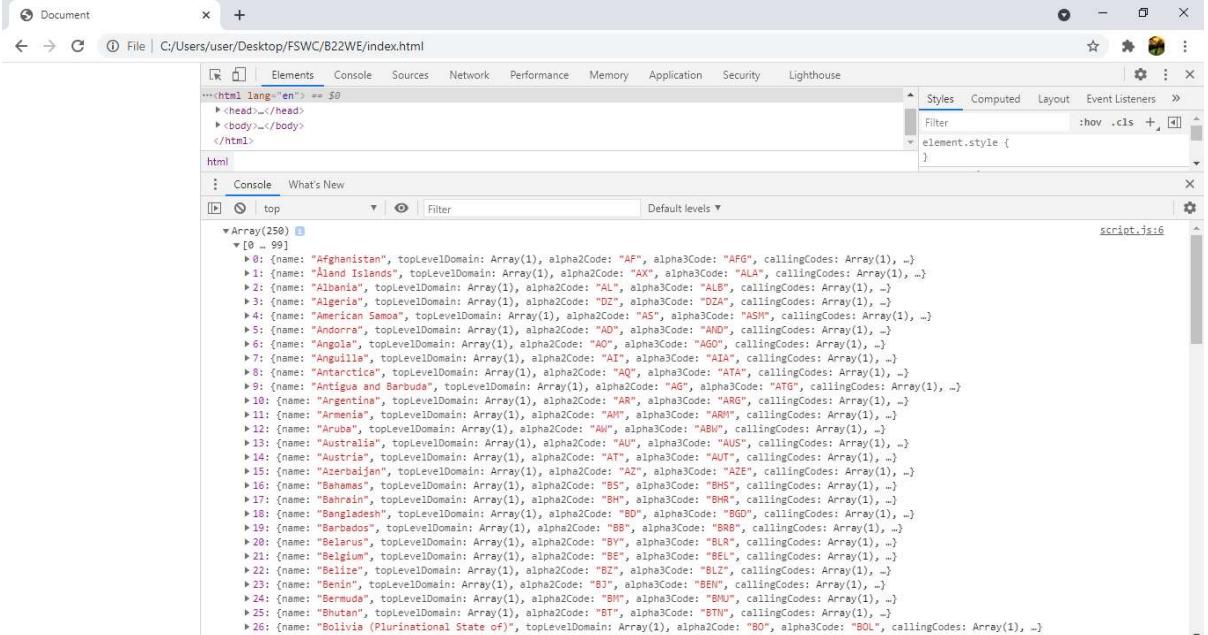
Script.js



```
File Edit Selection View Go Run Terminal Help
script.js - Untitled (Workspace) - Visual Studio Code

EXPLORER ... index.html JS script.js
OPEN EDITORS B22WE > JS script.js > onload
X index.html JS script.js
UNTITLED (WORKSPACE) B22WE
index.html JS script.js
1 var request=new XMLHttpRequest();
2 request.open('GET', 'https://restcountries.eu/rest/v2/all',true);
3 request.send()
4 request.onload=function(){
5     var data=JSON.parse(this.response);
6     console.log(data);
7 }
```

Output in console



```
Document + 
← → C File | C:/Users/user/Desktop/FSWC/B22WE/index.html
Elements Console Sources Network Performance Memory Application Security Lighthouse
HTML
Console What's New
Default levels
script.js:6
[0 .. 99]
0: {name: "Afghanistan", topLevelDomain: Array(1), alpha2Code: "AF", alpha3Code: "AFG", callingCodes: Array(1), ...}
1: {name: "Aland Islands", topLevelDomain: Array(1), alpha2Code: "AX", alpha3Code: "ALA", callingCodes: Array(1), ...}
2: {name: "Albania", topLevelDomain: Array(1), alpha2Code: "AL", alpha3Code: "ALB", callingCodes: Array(1), ...}
3: {name: "Algeria", topLevelDomain: Array(1), alpha2Code: "DZ", alpha3Code: "DZA", callingCodes: Array(1), ...}
4: {name: "American Samoa", topLevelDomain: Array(1), alpha2Code: "AS", alpha3Code: "ASM", callingCodes: Array(1), ...}
5: {name: "Andorra", topLevelDomain: Array(1), alpha2Code: "AD", alpha3Code: "AND", callingCodes: Array(1), ...}
6: {name: "Angola", topLevelDomain: Array(1), alpha2Code: "AO", alpha3Code: "AGO", callingCodes: Array(1), ...}
7: {name: "Anguilla", topLevelDomain: Array(1), alpha2Code: "AI", alpha3Code: "AIA", callingCodes: Array(1), ...}
8: {name: "Antarctica", topLevelDomain: Array(1), alpha2Code: "AQ", alpha3Code: "ATA", callingCodes: Array(1), ...}
9: {name: "Antigua and Barbuda", topLevelDomain: Array(1), alpha2Code: "AG", alpha3Code: "ATG", callingCodes: Array(1), ...}
10: {name: "Argentina", topLevelDomain: Array(1), alpha2Code: "AR", alpha3Code: "ARG", callingCodes: Array(1), ...}
11: {name: "Armenia", topLevelDomain: Array(1), alpha2Code: "AM", alpha3Code: "ARM", callingCodes: Array(1), ...}
12: {name: "Aruba", topLevelDomain: Array(1), alpha2Code: "AW", alpha3Code: "ABW", callingCodes: Array(1), ...}
13: {name: "Australia", topLevelDomain: Array(1), alpha2Code: "AU", alpha3Code: "AUS", callingCodes: Array(1), ...}
14: {name: "Austria", topLevelDomain: Array(1), alpha2Code: "AT", alpha3Code: "AUT", callingCodes: Array(1), ...}
15: {name: "Azerbaijan", topLevelDomain: Array(1), alpha2Code: "AZ", alpha3Code: "AZE", callingCodes: Array(1), ...}
16: {name: "Bahamas", topLevelDomain: Array(1), alpha2Code: "BS", alpha3Code: "BHS", callingCodes: Array(1), ...}
17: {name: "Bahrain", topLevelDomain: Array(1), alpha2Code: "BH", alpha3Code: "BHR", callingCodes: Array(1), ...}
18: {name: "Bangladesh", topLevelDomain: Array(1), alpha2Code: "BD", alpha3Code: "BGD", callingCodes: Array(1), ...}
19: {name: "Barbados", topLevelDomain: Array(1), alpha2Code: "BB", alpha3Code: "BRB", callingCodes: Array(1), ...}
20: {name: "Belarus", topLevelDomain: Array(1), alpha2Code: "BY", alpha3Code: "BLR", callingCodes: Array(1), ...}
21: {name: "Belgium", topLevelDomain: Array(1), alpha2Code: "BE", alpha3Code: "BEL", callingCodes: Array(1), ...}
22: {name: "Belize", topLevelDomain: Array(1), alpha2Code: "BZ", alpha3Code: "BLZ", callingCodes: Array(1), ...}
23: {name: "Benin", topLevelDomain: Array(1), alpha2Code: "BN", alpha3Code: "BEN", callingCodes: Array(1), ...}
24: {name: "Bermuda", topLevelDomain: Array(1), alpha2Code: "BM", alpha3Code: "BMU", callingCodes: Array(1), ...}
25: {name: "Bhutan", topLevelDomain: Array(1), alpha2Code: "BT", alpha3Code: "BTN", callingCodes: Array(1), ...}
26: {name: "Bolivia (Plurinational State of)", topLevelDomain: Array(1), alpha2Code: "BO", alpha3Code: "BOL", callingCodes: Array(1), ...}
```

b) to load only countries from rest countries API

index.html

The screenshot shows the Visual Studio Code interface with two open files. The left sidebar shows an 'EXPLORER' view with 'OPEN EDITORS' containing 'index.html' and 'script.js', and 'UNTITLED (WORKSPACE)' containing 'B22WE', 'index.html', and 'script.js'. The right pane shows the code for 'index.html':

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<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

The screenshot shows the Visual Studio Code interface with two open files. The left sidebar shows an 'EXPLORER' view with 'OPEN EDITORS' containing 'index.html' and 'script.js', and 'UNTITLED (WORKSPACE)' containing 'B22WE', 'index.html', and 'script.js'. The right pane shows the code for 'script.js':

```
var request=new XMLHttpRequest();
request.open('GET','https://restcountries.eu/rest/v2/all',true);
request.send()
request.onload=function(){
    var data=JSON.parse(this.response);
    for(var ele in data)
    {
        console.log(data[ele].name)
    }
}
```

Output on console

The screenshot shows a browser developer tools window with the 'Console' tab selected. The console output lists numerous country names:

```
Afghanistan
Åland Islands
Albania
Algeria
American Samoa
Andorra
Angola
Anguilla
Antarctica
Antigua and Barbuda
Argentina
Armenia
Aruba
Australia
Austria
Azerbaijan
Bahamas
Bahrain
Bangladesh
Barbados
Belarus
Belgium
Belize
Benin
```

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

SL.NO	Copy by value	Copy by reference
1.	Javascript has 5 data types that are passed by value : Boolean, null, undefined, String, and Number. These are primitive types .	Javascript has 3 data types that are passed by reference : Array, Function, and Object. These are all technically Objects, These are Non Primitive datatypes
2	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.	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.
3	Primitive data types are performed by copy by values	Non primitive data types are performed by copy by reference

Example of copy by value:

JavaScript
(known limitations)

```

1 var a=10;
2 var b=a;
3 a=20;
4 console.log(b)

```

[Edit this code](#)

line that just executed
next line to execute

<< First < Prev Next >> Last >>

Done running (4 steps)

Print output (drag lower right corner to resize)

10

Frames Objects

Global frame

a	20
b	10

Example of copy by reference:

JavaScript
(known limitations)

```

1 var arr1=[1,2,3];
2 var arr2=arr1
3 arr2[0]=10
4 console.log(arr2)
5 console.log(arr1)

```

[Edit this code](#)

line that just executed
next line to execute

<< First < Prev Next >> Last >>

Done running (5 steps)

[Customize visualization \(NEW!\)](#)

Print output (drag lower right corner to resize)

[10, 2, 3]
[10, 2, 3]

Frames Objects

Global frame

arr1	array						
arr2	<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><td>0</td><td>1</td><td>2</td></tr> <tr><td>10</td><td>2</td><td>3</td></tr> </table>	0	1	2	10	2	3
0	1	2					
10	2	3					

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

- Variables that are assigned a non-primitive value are given a *reference* to that value. That reference points to the object's location in memory. The variables don't actually contain the value.
- Objects are created at some location in the computer's memory. When we write `arr = []`, we've created an array in memory. What the variable `arr` receives is the address, the location, of that array.

For better understanding we will pretend that address is a new data type that is passed by value, just like number or string. An address points to the location, in memory, of a value that is passed by reference. Just like a string is denoted by quotation marks (" or ""), an address will be denoted by arrow brackets, `<>`.

When we assign and use a reference-type variable, what we write and see is:

- 1) `var arr = [];`
- 2) `arr.push(1);`

A representation of lines 1 and 2 above in memory is:

	Variables	Values	Addresses	Objects
1.	arr	<#001>	#001	[]
2.	arr	<#001>	#001	[1]

Notice that the value, the address, contained by the variable `arr` is static. The array in memory is what changes. When we use `arr` to do something, such as pushing a value, the Javascript engine goes to the location of `arr` in memory and works with the information stored there.

Assigning by Reference

When a reference type value, an object, is copied to another variable using `=`, the address of that value is what's actually copied over as if it were a primitive. Objects are copied by reference instead of by value.

```
var reference = [1];
var refCopy = reference;
```

The code above looks like this in memory

	Variables	Values	Addresses	Objects
	reference	<#001>	#001	[1]
	refCopy	<#001>		

Each variable now contains a reference to the *same array*. That means that if we alter reference, refCopy will see those changes:

```
reference.push(2);
console.log(reference, refCopy); // -> [1, 2], [1, 2]
```

Variables	Values	Addresses	Objects
reference	<#001>	#001	[1, 2]
refCopy	<#001>		

We've pushed 2 into the array in memory. When we use reference and refCopy, we're pointing to that same array.

Example:

JavaScript
(known limitations)

```
1 var arr1=[1,2,3];
2 var arr2=arr1
3 arr2[0]=10
4 console.log(arr2)
5 console.log(arr1)
```

[Edit this code](#)

line that just executed
next line to execute

<< First < Prev Next >> Last >>

Done running (5 steps)

Customize visualization (NEW!)

Print output (drag lower right corner to resize)

```
[ 10, 2, 3 ]
[ 10, 2, 3 ]
```

Frames Objects

Global frame

array

The screenshot shows a JavaScript debugger interface. On the left, there's a code editor with five lines of code. Line 1 creates an array arr1. Line 2 creates arr2 and assigns it the value of arr1. Line 3 changes the first element of arr2 to 10. Lines 4 and 5 log arr2 and arr1 respectively to the console. Below the code editor, there are buttons for navigating through the execution steps. To the right, there's a 'Print output' box showing the logs: '[10, 2, 3]' and '[10, 2, 3]'. Further down, there are tabs for 'Frames' and 'Objects'. Under 'Objects', there's a 'Global frame' section containing 'arr1' and 'arr2', each with a blue dot indicating they point to the same array object. To the right of these variables is a diagram of an array with indices 0, 1, 2, and 3, containing the values 10, 2, and 3. Arrows from both arr1 and arr2 point to the first element of the array.