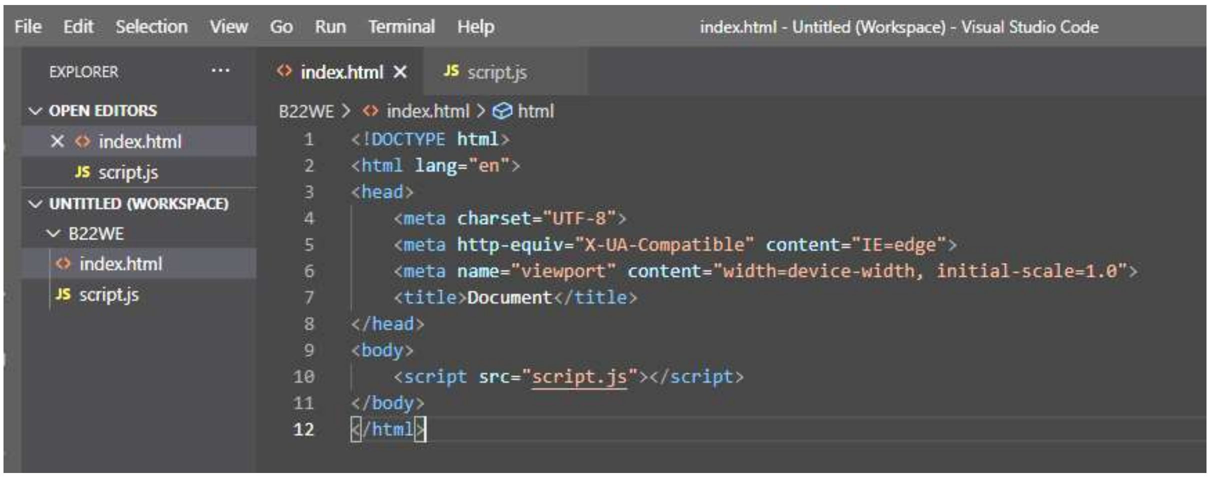
TASK 2- 20th March

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.

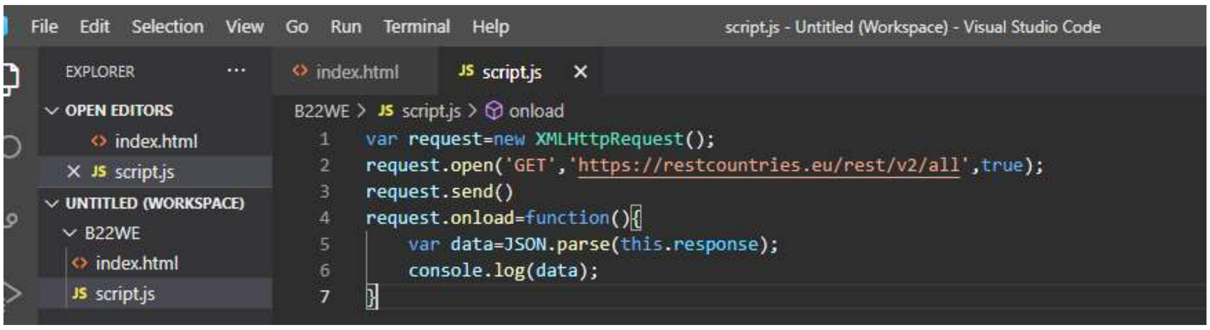
a) To load all the data of rest countries

Index.html



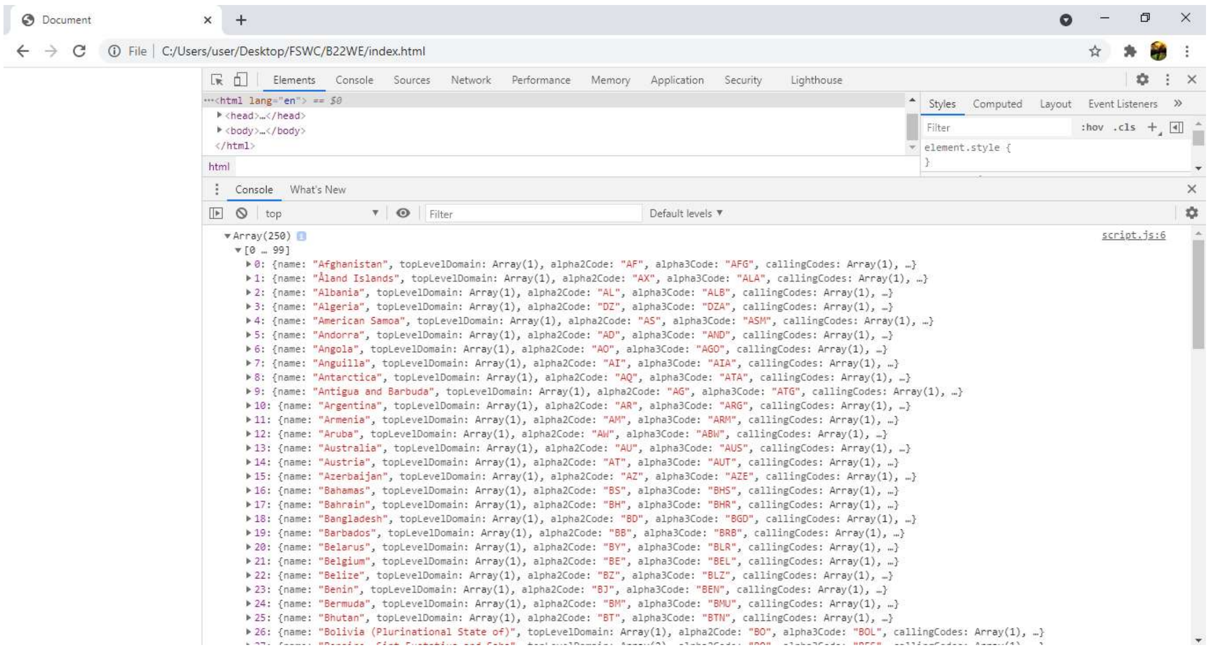
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script src="script.js"></script>
</body>
</html>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

Script.js



```javascript
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);
    console.log(data);
}
```

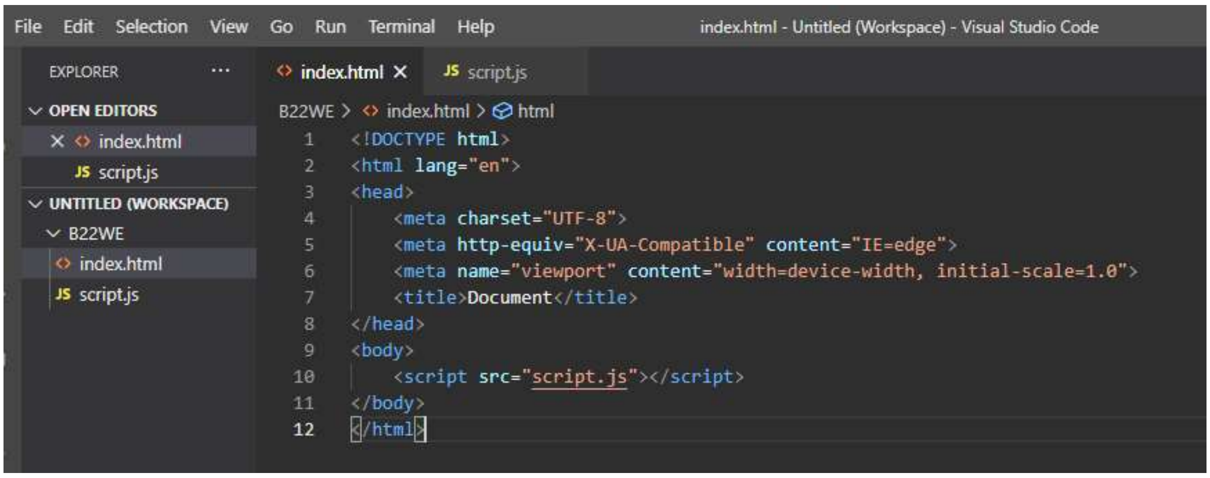
Output in console



```
▼Array(250)                                                                                          script.js:6
  ▼[0 … 99]
    ▶0: {name: "Afghanistan", topLevelDomain: Array(1), alpha2Code: "AF", alpha3Code: "AFG", callingCodes: Array(1), …}
    ▶1: {name: "Åland 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: "BJ", 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), …}
```

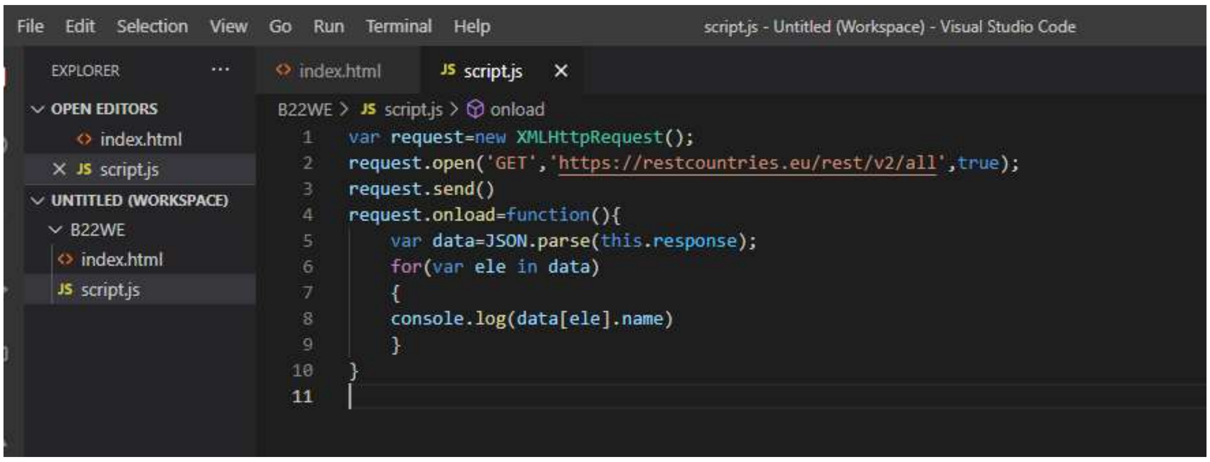
b) to load only countries from rest countries API

index.html



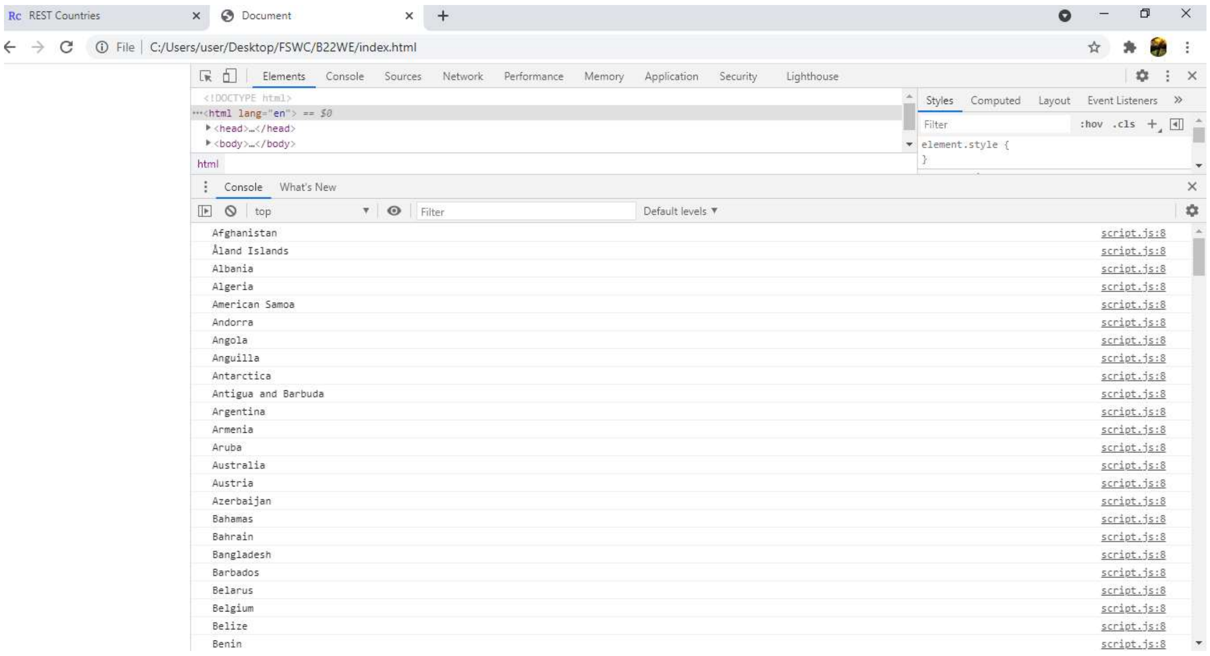
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script src="script.js"></script>
</body>
</html>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

Script.js



```javascript
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

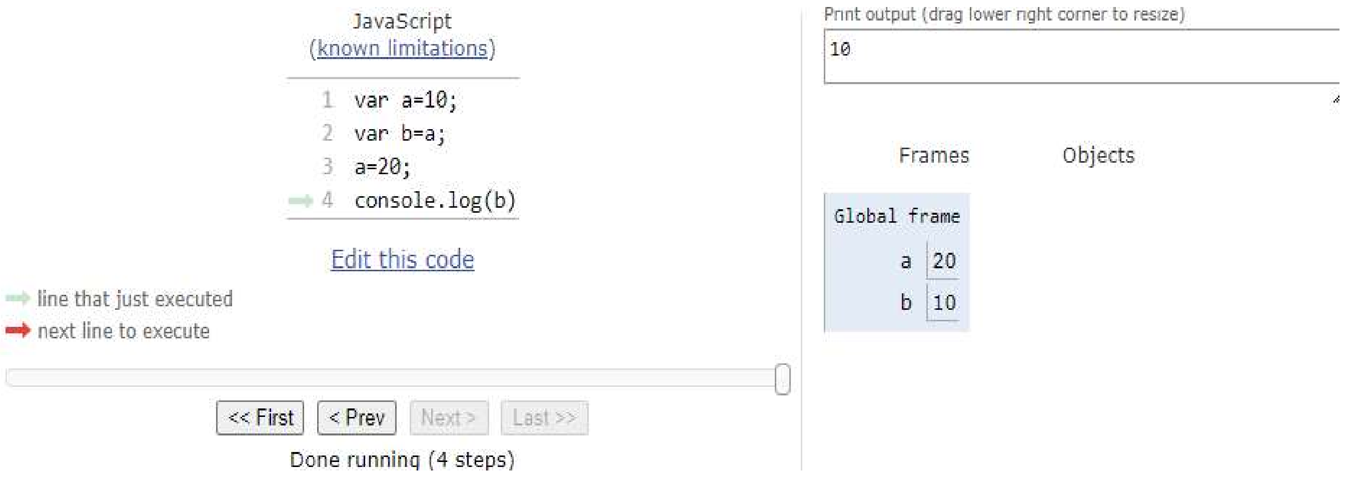


```
Afghanistan                 script.js:8
Åland Islands               script.js:8
Albania                     script.js:8
Algeria                     script.js:8
American Samoa              script.js:8
Andorra                     script.js:8
Angola                      script.js:8
Anguilla                    script.js:8
Antarctica                  script.js:8
Antigua and Barbuda         script.js:8
Argentina                   script.js:8
Armenia                     script.js:8
Aruba                       script.js:8
Australia                   script.js:8
Austria                     script.js:8
Azerbaijan                  script.js:8
Bahamas                     script.js:8
Bahrain                     script.js:8
Bangladesh                  script.js:8
Barbados                    script.js:8
Belarus                     script.js:8
Belgium                     script.js:8
Belize                      script.js:8
Benin                       script.js:8
```

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:



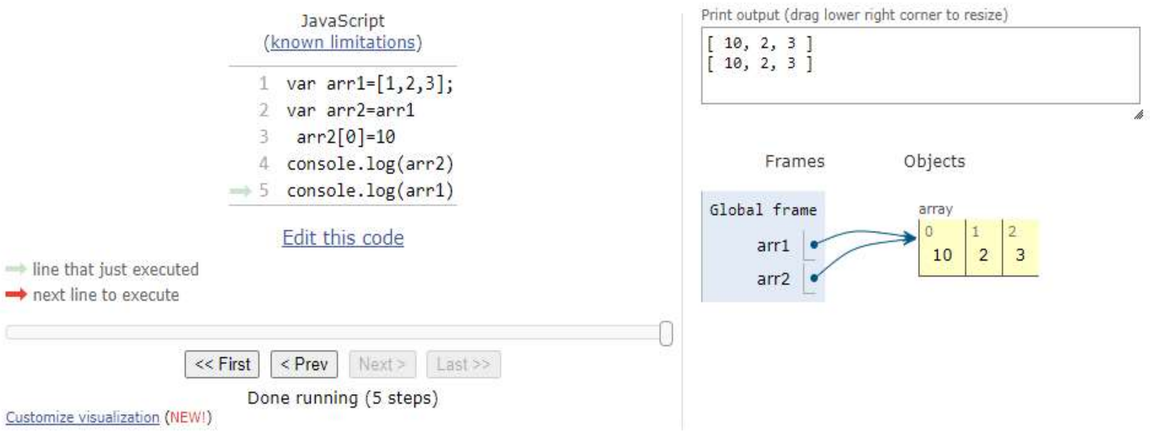
```
var a=10;
var b=a;
a=20;
console.log(b)
```

Done running (4 steps)

```
10
```

Global frame	
a	20
b	10

Example of copy by reference:



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

Done running (5 steps)

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

Global frame: arr1 → array, arr2 → array

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:

1.

Variables	Values	Addresses	Objects
arr	<#001>	#001	[]

2.

Variables	Values	Addresses	Objects
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>		

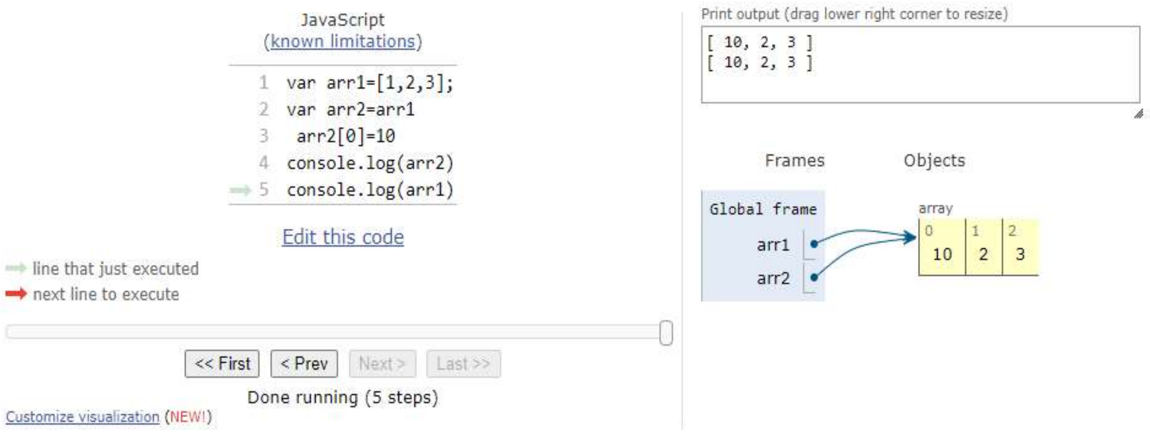
Each variable now contains a reference to the *same array*. That means that if we alter reference, refCopy will see those changes:

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:



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

Print output (drag lower right corner to resize)

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