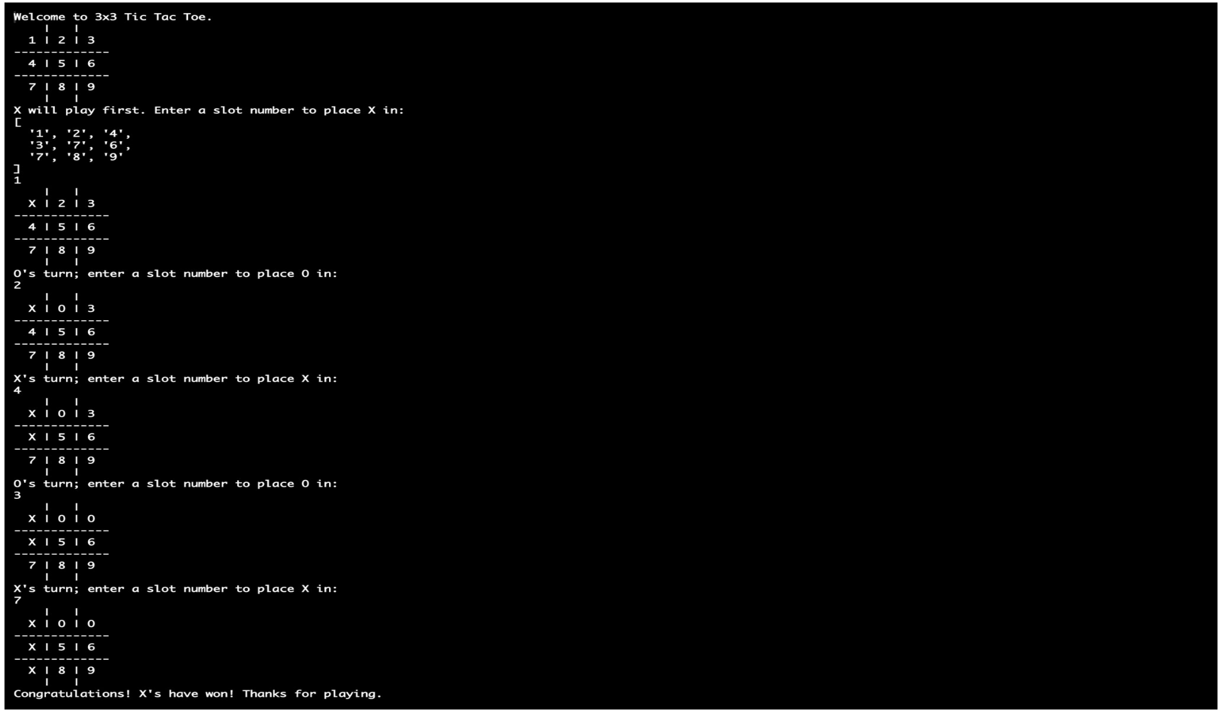
App : Tic-Tac-Toe App
Arrays, Strings, Objects, JSON, Destructuring, forEach(), map(), filter() and Rest parameters in Javascript

Relevel
by Unacademy

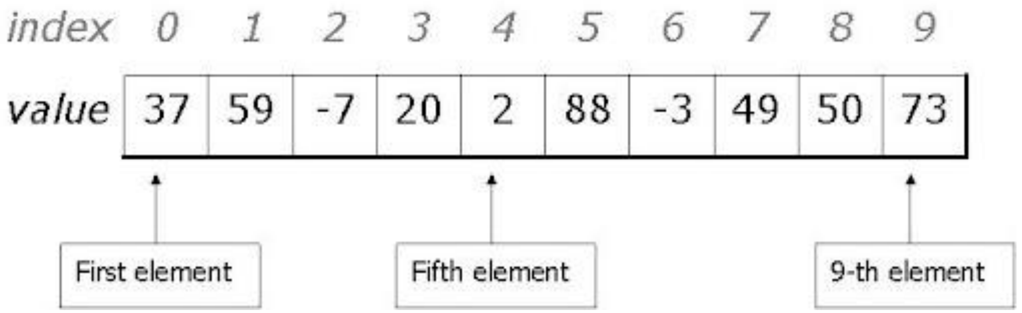
Tic-Tac-Toe App - How it could be designed ?



```
Welcome to 3x3 Tic Tac Toe.
    |   |
  1 | 2 | 3
-------------
  4 | 5 | 6
-------------
  7 | 8 | 9
    |   |
X will play first. Enter a slot number to place X in:
[
  '1', '2', '4',
  '3', '7', '6',
  '7', '8', '9'
]
1
    |   |
  X | 2 | 3
-------------
  4 | 5 | 6
-------------
  7 | 8 | 9
    |   |
O's turn; enter a slot number to place O in:
2
    |   |
  X | O | 3
-------------
  4 | 5 | 6
-------------
  7 | 8 | 9
    |   |
X's turn; enter a slot number to place X in:
4
    |   |
  X | O | 3
-------------
  X | 5 | 6
-------------
  7 | 8 | 9
    |   |
O's turn; enter a slot number to place O in:
3
    |   |
  X | O | O
-------------
  X | 5 | 6
-------------
  7 | 8 | 9
    |   |
X's turn; enter a slot number to place X in:
7
    |   |
  X | O | O
-------------
  X | 5 | 6
-------------
  X | 8 | 9
    |   |
Congratulations! X's have won! Thanks for playing.
```

Arrays

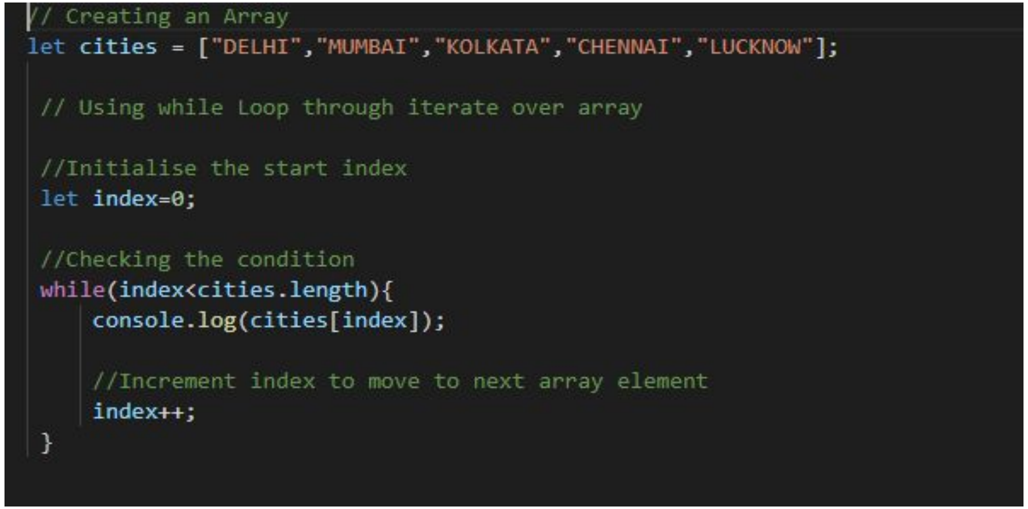
An *array* is an ordered list of values which is used to store data in linear fashion



Arrays

Declare + Initialize Array :

let array_names = [“ABC”,”BCD”,CDF”,THR”]



```
// Creating an Array
let cities = ["DELHI","MUMBAI","KOLKATA","CHENNAI","LUCKNOW"];

 // Using while Loop through iterate over array

 //Initialise the start index
 let index=0;

 //Checking the condition
 while(index<cities.length){
     console.log(cities[index]);

     //Increment index to move to next array element
     index++;
 }
```

Arrays

Types of Array

Homogeneous Arrays : In homogenous array only single type of data is stored in array

```
let array = ["Matthew", "Simon", "Luke"];

let array = [27, 24, 30];
```

Heterogeneous Arrays : In heterogeneous array mixed data types can be used to store in array

```
let array = [“ABC”,123,true,1]
```

Arrays

Types of Array

Multi-dimensional Arrays : An array which has an array stored within another array is called multidimensional array

```
var array = [["Matthew", "27"], ["Simon", "24"], ["Luke", "30"]];
```

Jagged Arrays : Jagged arrays are similar to multidimensional array with the exception being that a jagged array does not require a uniform set of data.

```
let array = [
      ["Matthew", "27", "Developer"],
       ["Simon", "24"],
        ["Luke"]
  ];
```

Strings

Strings are sequence of characters that are enclosed in single or double quotes
Eg : “Javascript”,”React js”,”Node js”

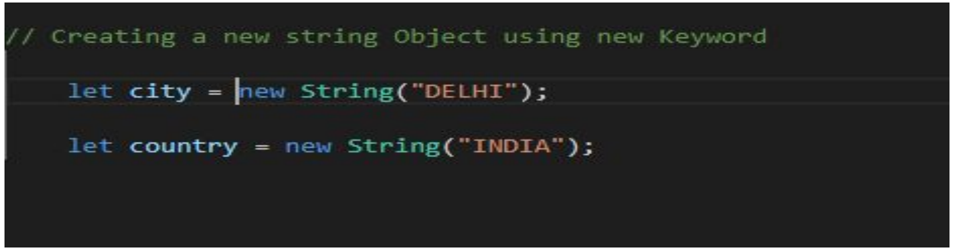
How to create Strings in Js :

String Literals -

The easiest and the mostcommon way of creating a String is using String Literals .Eg: Let city =”NEW DELHI”

New Keyboard -

Using the Javascript new keyword we can create a new String object .Eg :

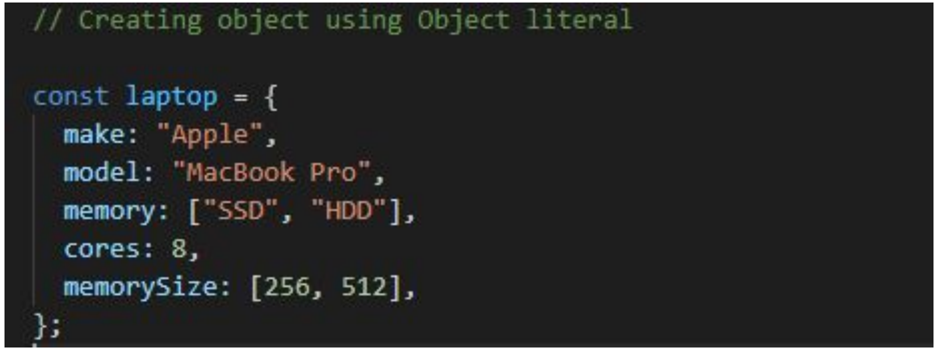


```
// Creating a new string Object using new Keyword

    let city = new String("DELHI");

    let country = new String("INDIA");
```

Objects in Js

Object in Javascript is an entity which has properties and methods associated with it. These objects can have properties in the form of “key : value” pairs, here key is a string which can also be referred to as property name and value can be anything (e.g. string, number, null, array, function etc.).

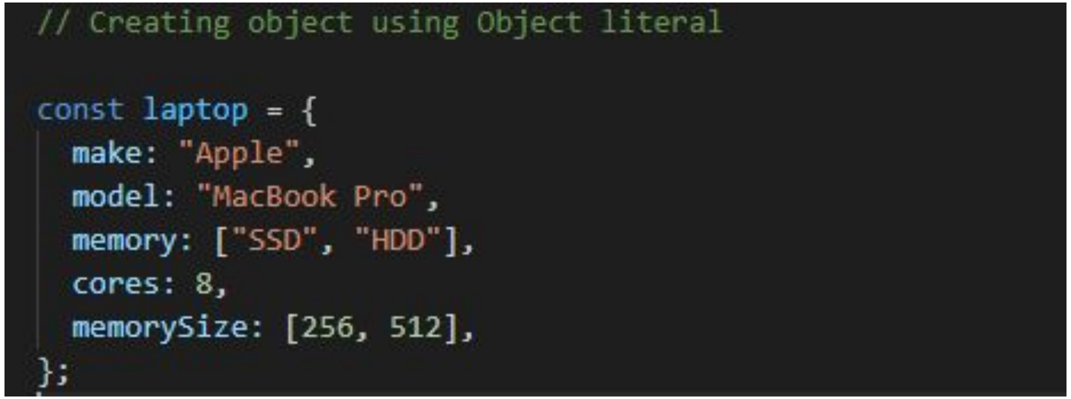
Everything in Javascript is an object except the primitive data types that include number, string, boolean, null, undefined.



```
// Creating object using Object literal

const laptop = {
  make: "Apple",
  model: "MacBook Pro",
  memory: ["SSD", "HDD"],
  cores: 8,
  memorySize: [256, 512],
};
```

Creating Objects using Object literal

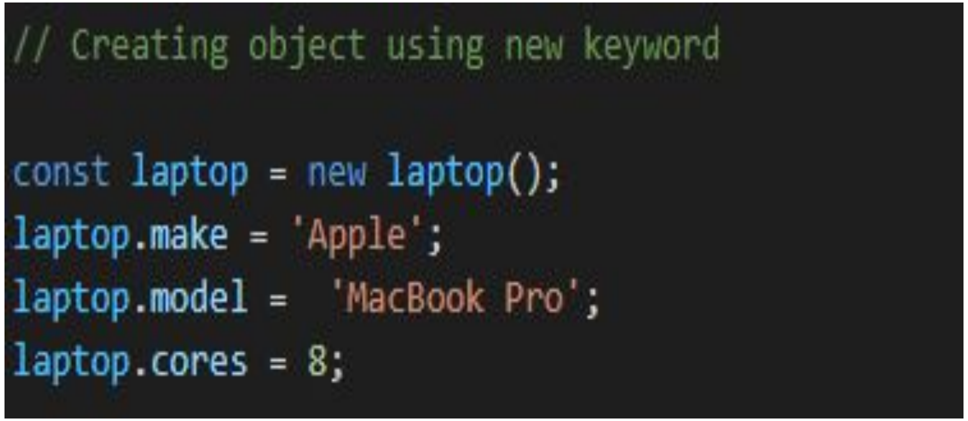
1. The easiest and the most common way of creating an object is using Object Literals



```
// Creating object using Object literal

const laptop = {
  make: "Apple",
  model: "MacBook Pro",
  memory: ["SSD", "HDD"],
  cores: 8,
  memorySize: [256, 512],
};
```

Creating Objects using New Keyword

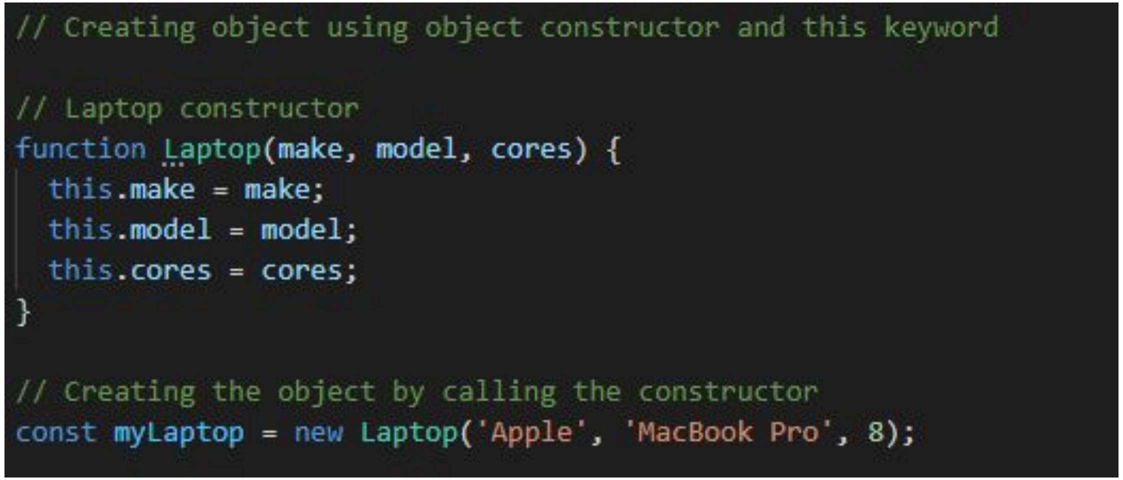
2. Using the new Javascript keyword, we can create a new object.



```
// Creating object using new keyword

const laptop = new laptop();
laptop.make = 'Apple';
laptop.model =  'MacBook Pro';
laptop.cores = 8;
```

Creating Objects using Constructor and this keyword

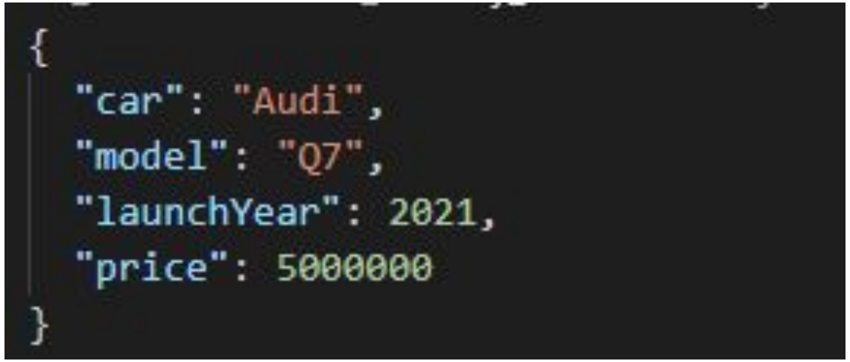
Using this keyword and object constructor, we can create a new object.



```
// Creating object using object constructor and this keyword

// Laptop constructor
function Laptop(make, model, cores) {
  this.make = make;
  this.model = model;
  this.cores = cores;
}

// Creating the object by calling the constructor
const myLaptop = new Laptop('Apple', 'MacBook Pro', 8);
```

JSON JavaScript Object Notation

JSON stands for JavaScript Object Notation. It is basically a widely accepted format to exchange data between various application including client-server
applications as well and also a great alternative to XML. The file containing JSON objects is saved with the extension .json.



```
{
    "car": "Audi",
    "model": "Q7",
    "launchYear": 2021,
    "price": 5000000
}
```

Difference in JSON Object and Javascript Object:

Though JSON Object and Javascript Object have a close resemblance as both are key:value pairs there are some things we need to note

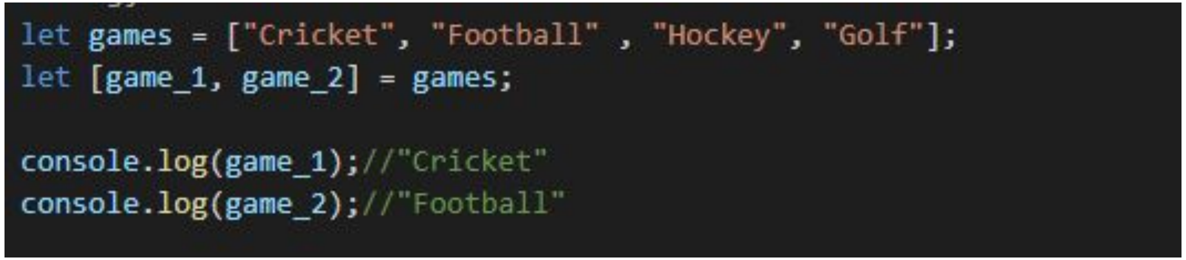
Property name / key are always in double quotes " "

The keys are any valid string but the JSON values can only be one of the six data types (strings, numbers, objects, arrays, boolean, null). Unlike in Javascript Objects the values can literally be anything as we saw earlier.

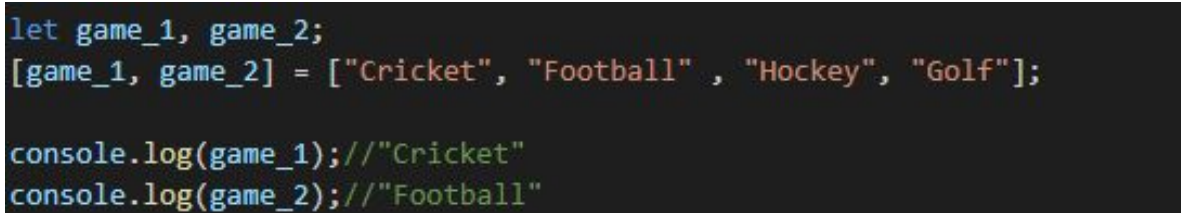
Destructuring in Javascript:

Destructuring in JavaScript is an expression that makes it possible to unpack values from arrays, or properties from objects. We can extract data from arrays and objects and assign them to distinct variables.The value that should be unpacked from the sourced variable is defined on the left-hand side.

Array Destructuring

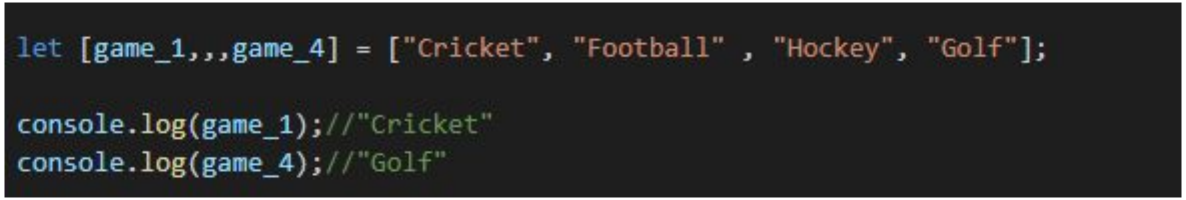


```
let games = ["Cricket", "Football" , "Hockey", "Golf"];
let [game_1, game_2] = games;

console.log(game_1);//"Cricket"
console.log(game_2);//"Football"
```



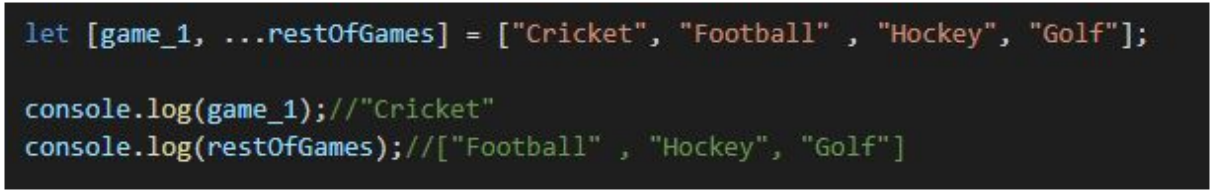
```
let game_1, game_2;
[game_1, game_2] = ["Cricket", "Football" , "Hockey", "Golf"];

console.log(game_1);//"Cricket"
console.log(game_2);//"Football"
```

Skipping Elements in an Array:



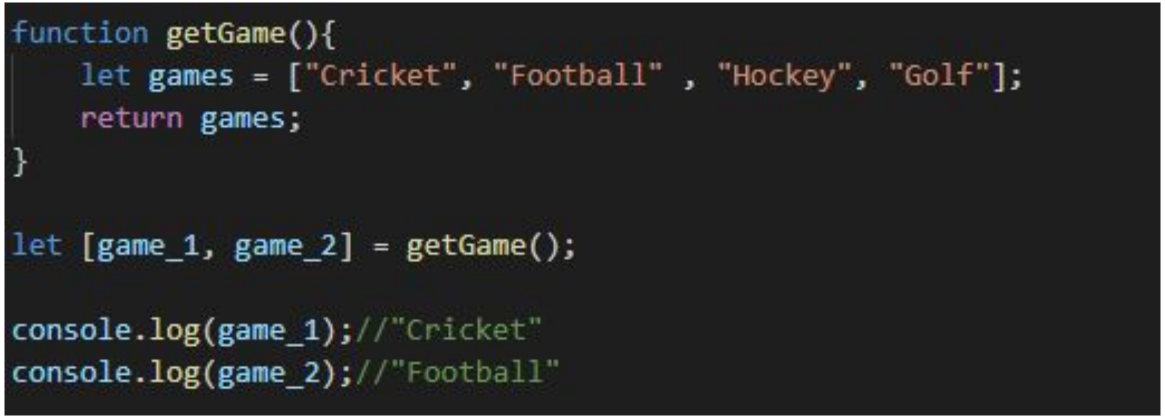
```
let [game_1,,,game_4] = ["Cricket", "Football" , "Hockey", "Golf"];

console.log(game_1);//"Cricket"
console.log(game_4);//"Golf"
```

Assigning rest of the Array:



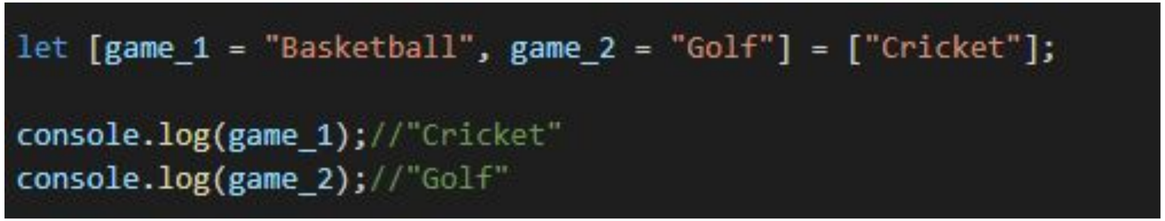
```
let [game_1, ...restOfGames] = ["Cricket", "Football" , "Hockey", "Golf"];

console.log(game_1);//"Cricket"
console.log(restOfGames);//["Football" , "Hockey", "Golf"]
```

Destructuring assignment Using Functions:



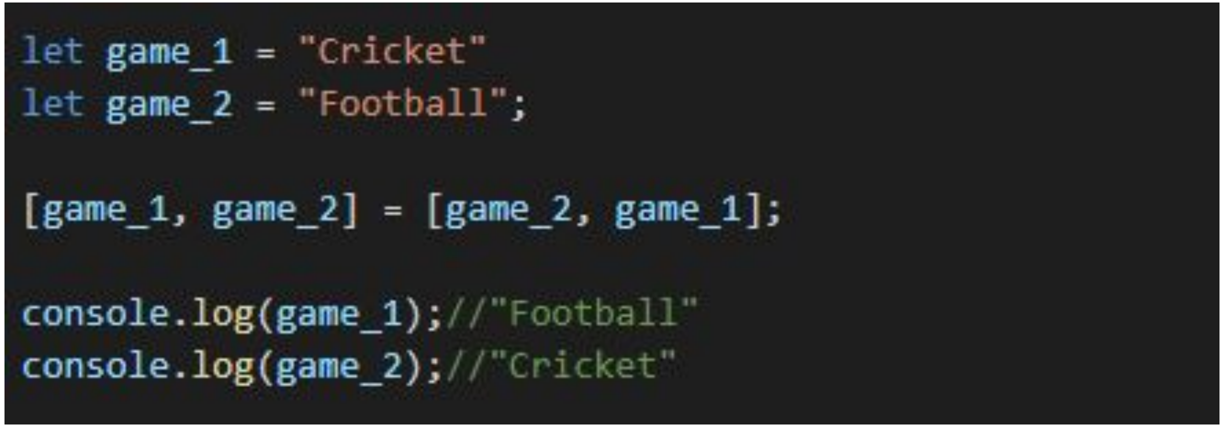
```
function getGame(){
    let games = ["Cricket", "Football" , "Hockey", "Golf"];
    return games;
}

let [game_1, game_2] = getGame();

console.log(game_1);//"Cricket"
console.log(game_2);//"Football"
```

Using Default Values:



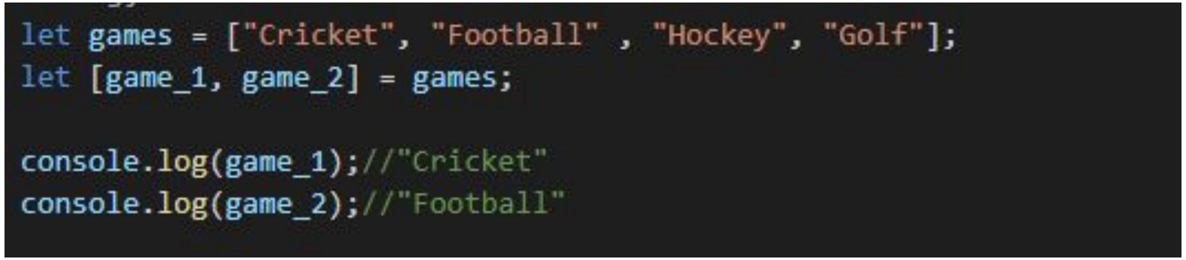
```
let [game_1 = "Basketball", game_2 = "Golf"] = ["Cricket"];

console.log(game_1);//"Cricket"
console.log(game_2);//"Golf"
```

Swapping values using Destructuring assignment:

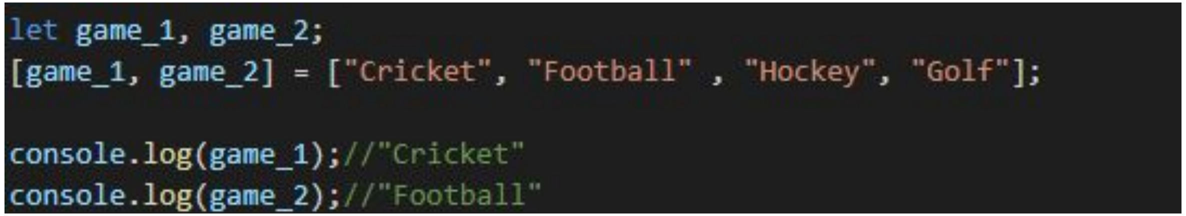


```
let game_1 = "Cricket"
let game_2 = "Football";

[game_1, game_2] = [game_2, game_1];

console.log(game_1);//"Football"
console.log(game_2);//"Cricket"
```

Array Destructuring

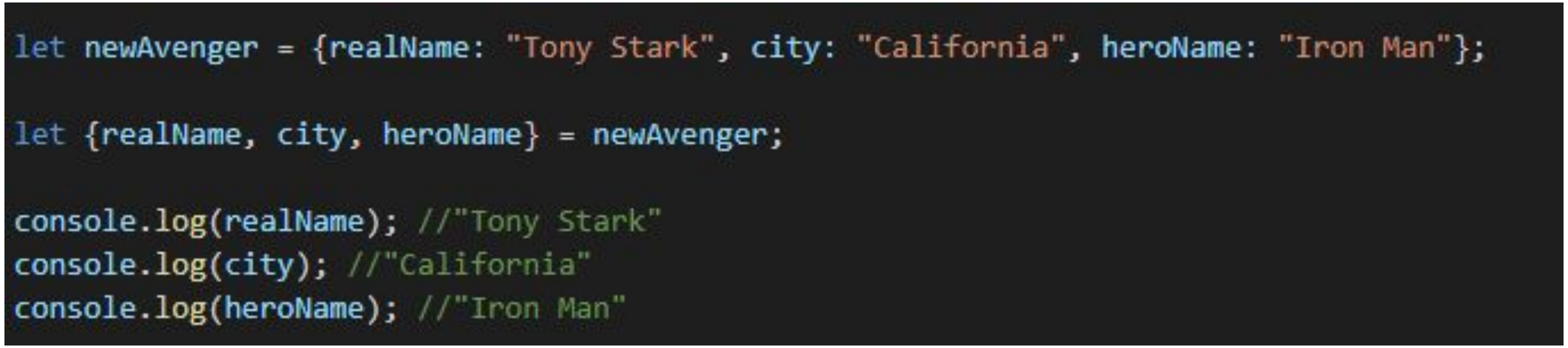


```
let games = ["Cricket", "Football" , "Hockey", "Golf"];
let [game_1, game_2] = games;

console.log(game_1);//"Cricket"
console.log(game_2);//"Football"
```



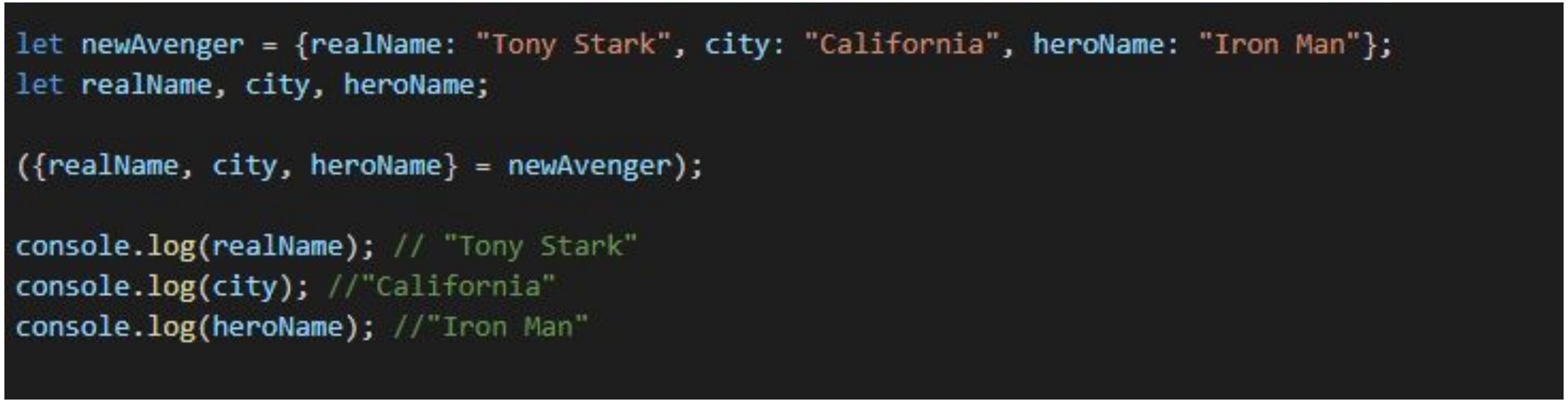
```
let game_1, game_2;
[game_1, game_2] = ["Cricket", "Football" , "Hockey", "Golf"];

console.log(game_1);//"Cricket"
console.log(game_2);//"Football"
```

Object Destructuring



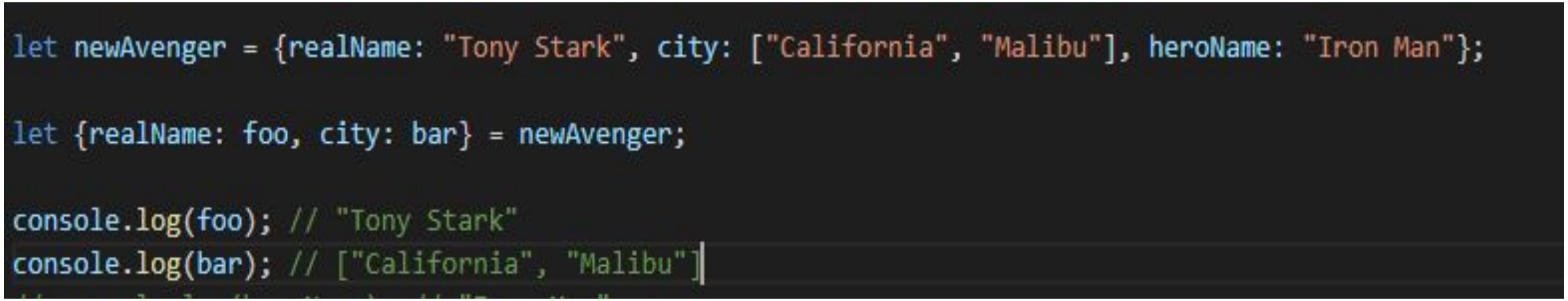
```
let newAvenger = {realName: "Tony Stark", city: "California", heroName: "Iron Man"};

let {realName, city, heroName} = newAvenger;

console.log(realName); //"Tony Stark"
console.log(city); //"California"
console.log(heroName); //"Iron Man"
```

Declaring the variables before destructuring assignment



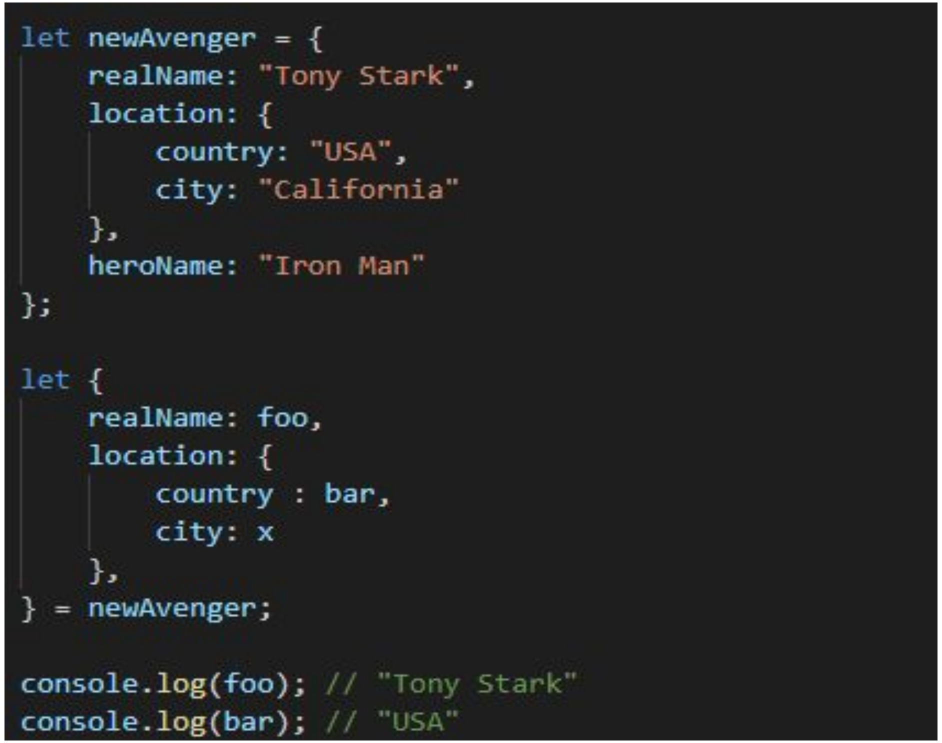
```
let newAvenger = {realName: "Tony Stark", city: "California", heroName: "Iron Man"};
let realName, city, heroName;

({realName, city, heroName} = newAvenger);

console.log(realName); // "Tony Stark"
console.log(city); //"California"
console.log(heroName); //"Iron Man"
```

Combination of Array and Object Destructuring:



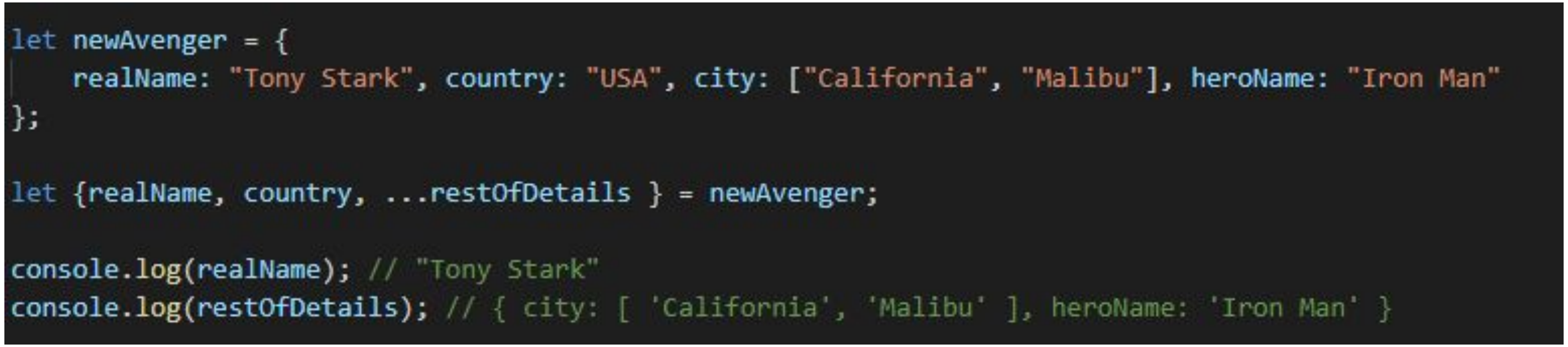
```
let newAvenger = {realName: "Tony Stark", city: ["California", "Malibu"], heroName: "Iron Man"};

let {realName: foo, city: bar} = newAvenger;

console.log(foo); // "Tony Stark"
console.log(bar); // ["California", "Malibu"]
```

Object Destructuring in Nested Objects:



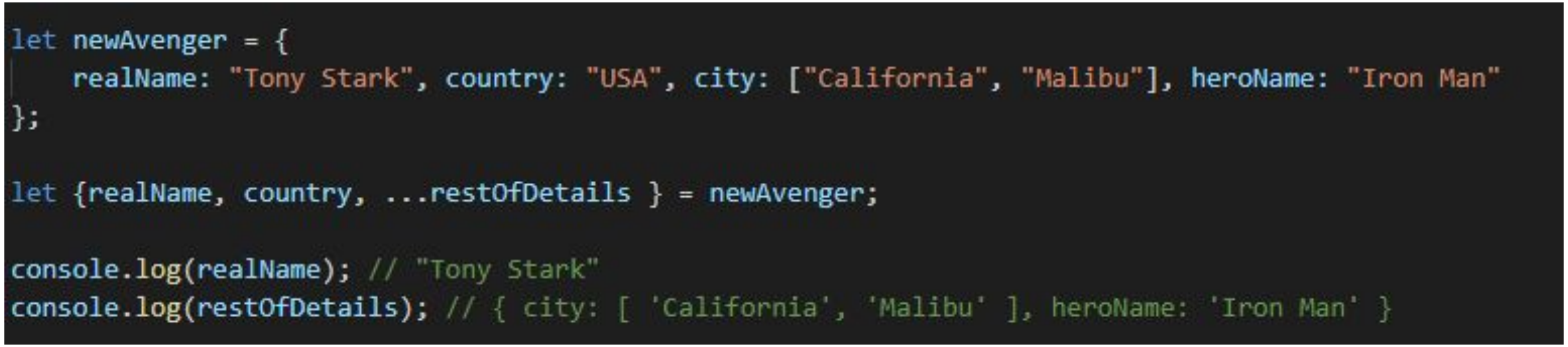
```
let newAvenger = {
    realName: "Tony Stark",
    location: {
        country: "USA",
        city: "California"
    },
    heroName: "Iron Man"
};

let {
    realName: foo,
    location: {
        country : bar,
        city: x
    },
} = newAvenger;

console.log(foo); // "Tony Stark"
console.log(bar); // "USA"
```

Rest in Object Destructuring:



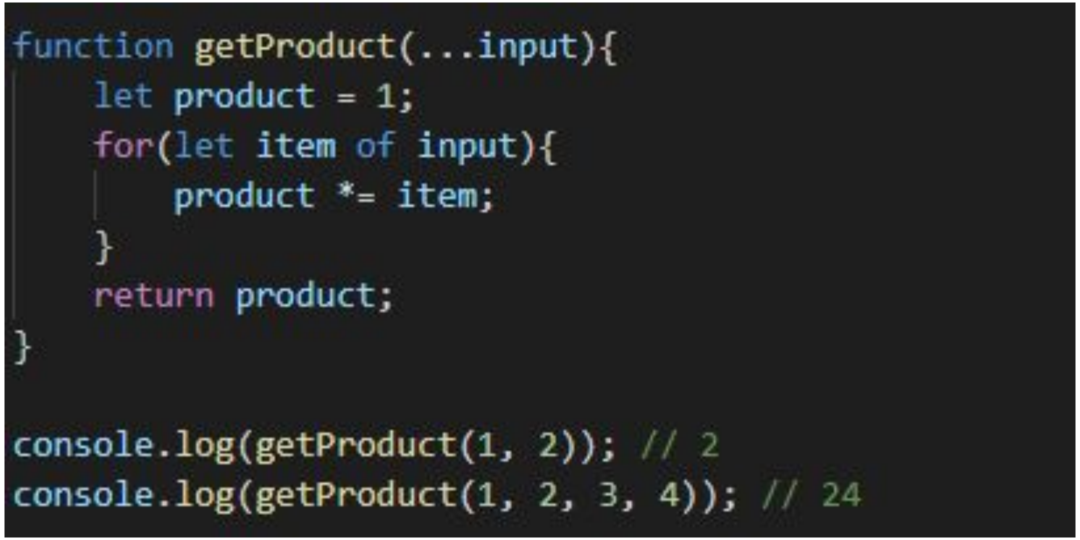
```
let newAvenger = {
    realName: "Tony Stark", country: "USA", city: ["California", "Malibu"], heroName: "Iron Man"
};

let {realName, country, ...restOfDetails } = newAvenger;

console.log(realName); // "Tony Stark"
console.log(restOfDetails); // { city: [ 'California', 'Malibu' ], heroName: 'Iron Man' }
```

Rest in Object Destructuring:



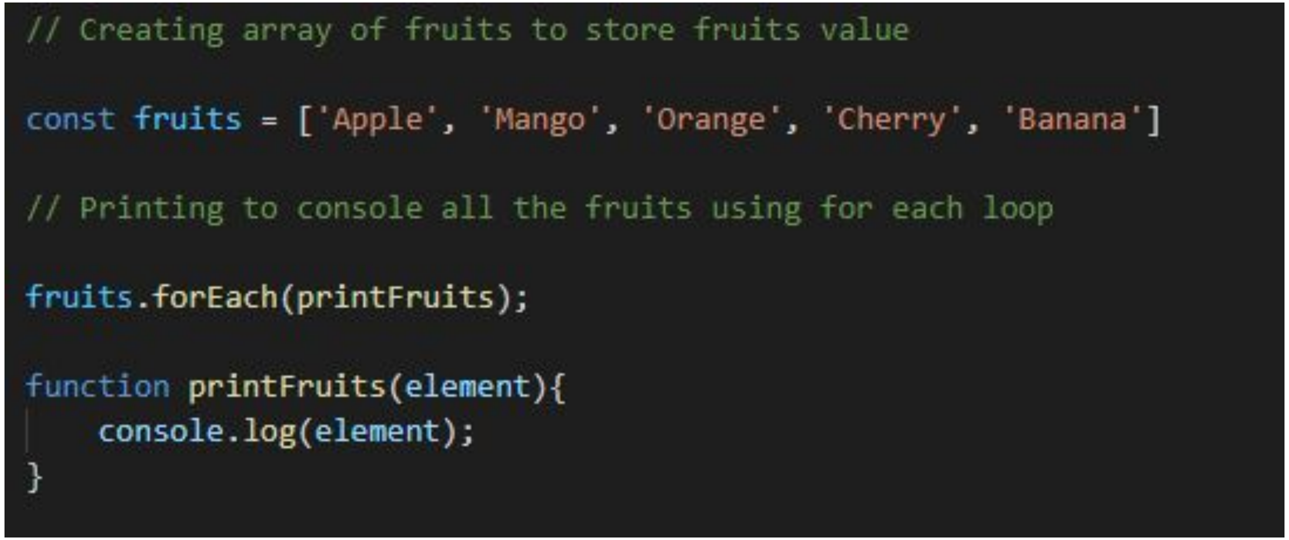
```
let newAvenger = {
    realName: "Tony Stark", country: "USA", city: ["California", "Malibu"], heroName: "Iron Man"
};

let {realName, country, ...restOfDetails } = newAvenger;

console.log(realName); // "Tony Stark"
console.log(restOfDetails); // { city: [ 'California', 'Malibu' ], heroName: 'Iron Man' }
```

Rest Parameters:



```
function getProduct(...input){
    let product = 1;
    for(let item of input){
        product *= item;
    }
    return product;
}

console.log(getProduct(1, 2)); // 2
console.log(getProduct(1, 2, 3, 4)); // 24
```

For Each() Loop:



```
// Creating array of fruits to store fruits value

const fruits = ['Apple', 'Mango', 'Orange', 'Cherry', 'Banana']

// Printing to console all the fruits using for each loop

fruits.forEach(printFruits);

function printFruits(element){
    console.log(element);
}
```

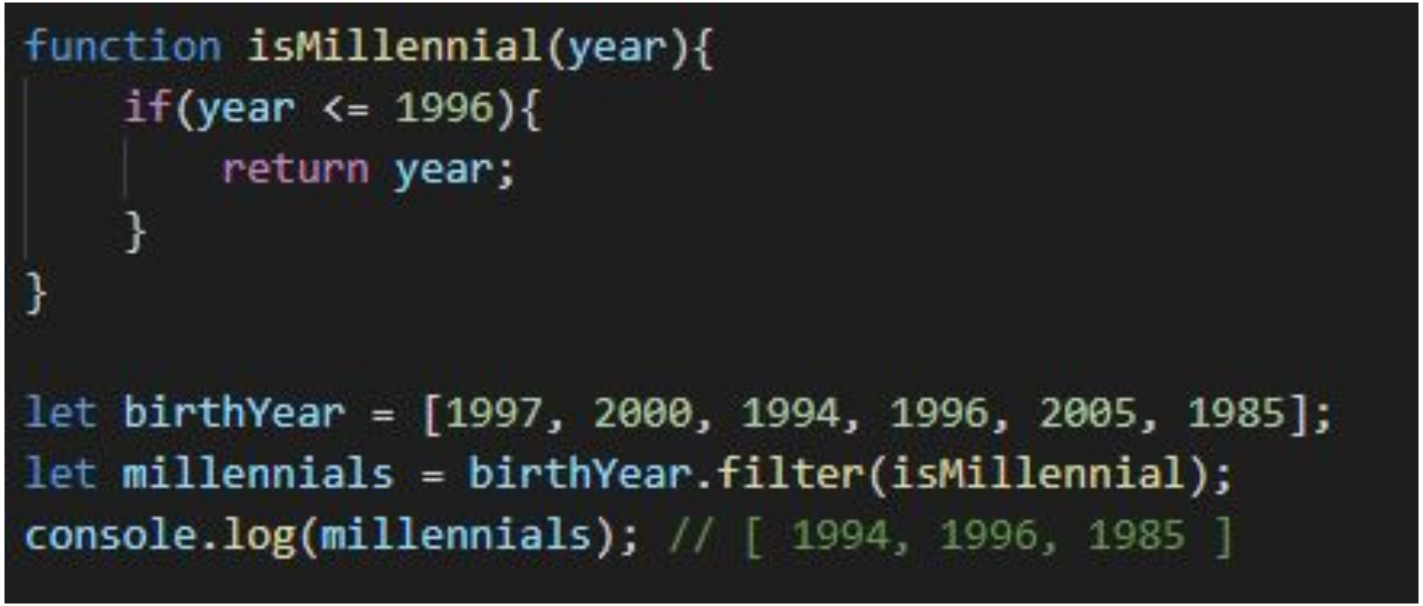
filter() method:

filter() method creates a new array from a given array which consists of only those elements from the given array which satisfy a certain test/condition.

Syntax:

array.filter(callback(element, index, arr), thisArg)

Example



```
function isMillennial(year){
    if(year <= 1996){
        return year;
    }
}

let birthYear = [1997, 2000, 1994, 1996, 2005, 1985];
let millennials = birthYear.filter(isMillennial);
console.log(millennials); // [ 1994, 1996, 1985 ]
```

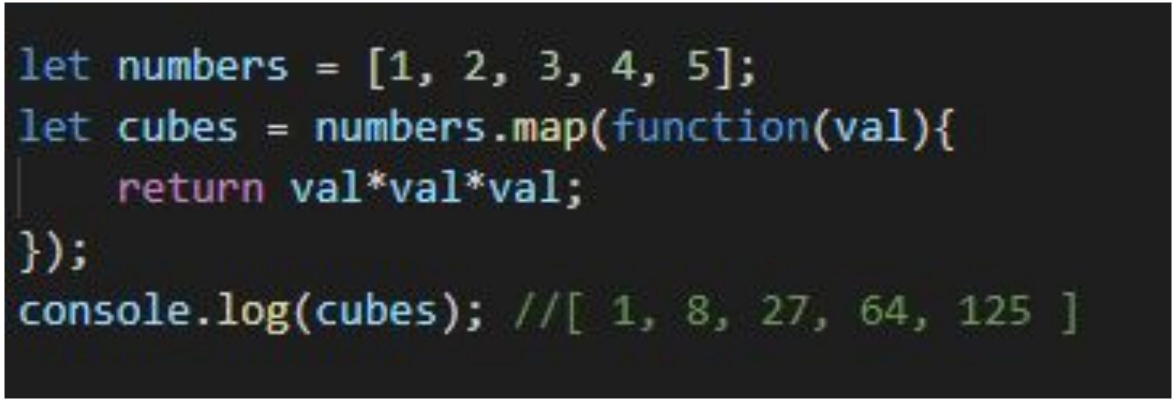
map() method:

map() method applies a function, performs the desired operation on each element of the given array, and returns a new array consisting of all the elements after applying the function. map() allows elements of the array to be manipulated as per the user's preference.

Syntax:

```
array.filter(callback(element, index, arr), thisArg)
```

Example



```
let numbers = [1, 2, 3, 4, 5];
let cubes = numbers.map(function(val){
    return val*val*val;
});
console.log(cubes); //[ 1, 8, 27, 64, 125 ]
```

Thank you