R.Preethi
29/05/2023

JAVASCRIPT

Definition:
Javascript is a scripting language used to create and control dynamic web pages. Also it is used from server-side and client-side in your application.

Comments:
JavaScript comments can be used to explain code and to make it more readable.

// Single line comments
/* some line */ Multiline comments.

Variables → container for storing the data

There are three types of variables

- var:
  - ➔ It supports the old version
  - ➔ Acts as a global scope access from outside of the scope.
  - ➔ Re-declare the variable. It Access reassign the value

  Example:

```
var a= 33;       // declare the variable
var b= 5;        // declare the variable
consle.log (a+b)    // 38 output.
```

- let:
  - ➔ Only use let if you can't use const.
  - ➔ Acts as a local scope doesn't access from outside of the scope.
  - ➔ Once you can declare a variable it does not redeclare but it access redeclares a variable outside the block.
  - ➔ It Access reassign the value

  Example:

```
let name = "Preethi"   // declare the variable
consle.log(name)       // output.
```

- const:
  - ➔ Declare a constant value we have to use const and its same as let.
  - ➔ But do not not reassign the value, when creating an object we can reassign the value.
  - ➔ When use const:

- new Array
- new object
- new function
- new RegExp

Example:

```
const price 1 = 100,
const price2 =200,
consle.log(price1 + price2) // output 300
```

Operators

Performing specific operations between two operands(number). There are different types of operators such as

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators
- Ternary Operators

A. Arithmetic Operators

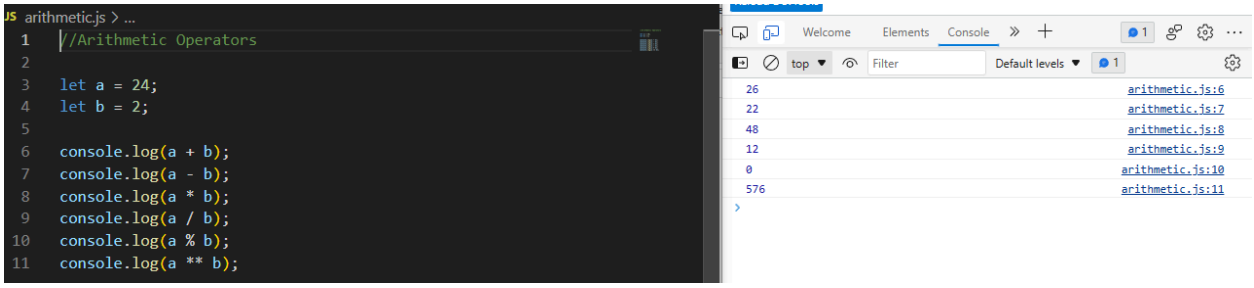
Operators are used to perform arithmetic on numbers

- Addition → Add the numbers(+)
- Subtraction → Subtract the number(-)
- Multiplication → Multiply the numbers(*)
- Division → Divide the numbers (/)
- Modulus → Get the remainder of number(%)
- Exponentiation → Power of the number (**)

Example:

```
Let a =24;
Let b = 2;
console.log(a+b);        // addition
console.log(a-b);        // subtraction
```

```
console.log(a*b);       // multiplication
console.log(a/b);       // division
console.log(a%b);       // modulus
console.log (a * b)     // exponentiation
```



B. Assignment Operators:

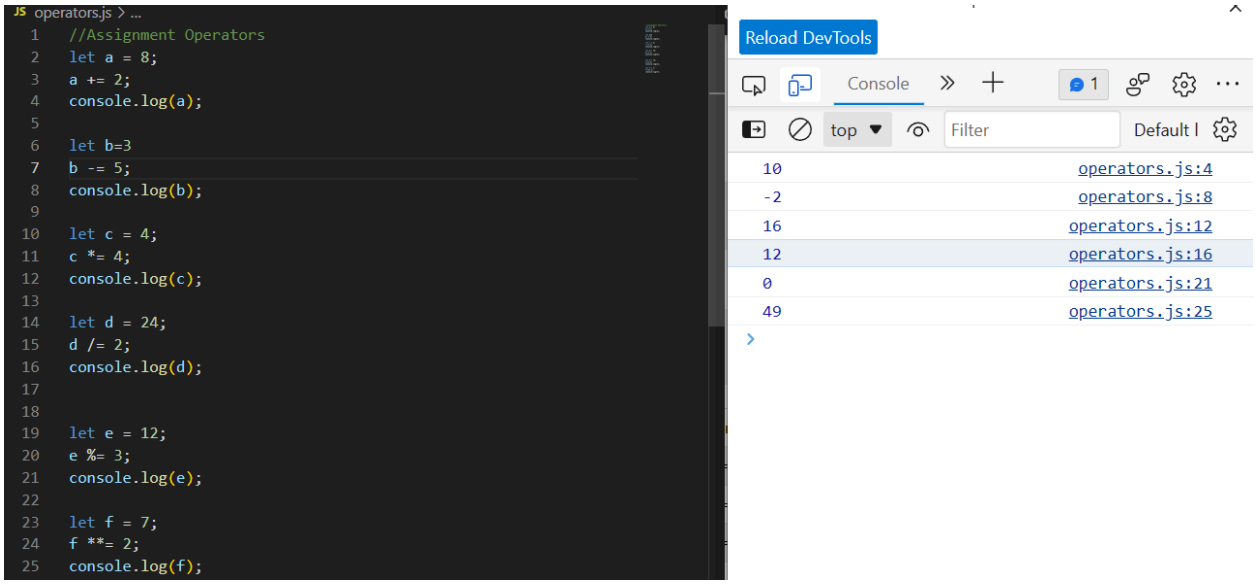
To assign the value for increment and decrement the values.

- = → assign the value
- += → adds a value to a variable
- -= → subtracts a value to a variable.
- *= → Multiply a value to a variable.
- /= → Divide a value
- %= → assigns a remainder to a variable.
- **= → raises a variable to the power of the operand.

Example:

```
let a = 8;
a += 2;
a= a + 2;          // add a value
console.log(a);

let b = 3;
b -= 5             // subtracts the value
console.log(b);

let c = 4;
c *= 4             // multiplies the value
console.log(c);
```

```
let d = 24;
d /= 2;            // divide the value
console.log(d);

let e = 12;
e %= 3;            // modulus a value
console.log(e);

let f = 7;
f **= 2;           // exponentiation
console.log(f);
```



C. Comparison Operators:

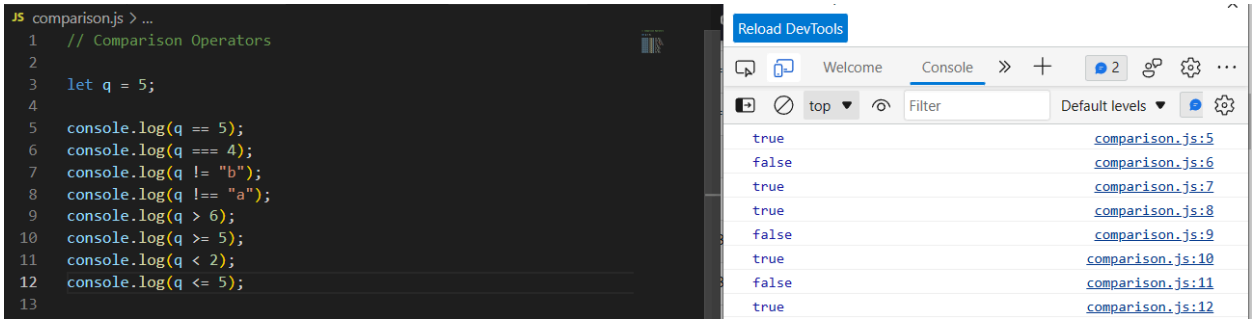
Comparison operators are used in logical statements to determine equality or difference between variables or values.

- == → compare the values of a variable.
- === → compare both the values and type of a variable
- != → not equal
- !== → not equal value or not equal type
- > → greater than
- >= → greater than or equal

- < → less than
- <= → less than or equal

Example:

```
Let q = 5;
console.log(q == 5);     // equals the value
console.log(q == 4);     // equals both the value and datatype
console.log(5 != “b”);   // not equal to
console.log(5 !== “a”);// not equal to both the sides
console.log(6 > 6);      // greater than
console.log(q >= 5);     // greater than or equal to
console.log(q < 2);      // less than
console.log(q <= 5);     // less than or equal to
```



D. Logical Operator:

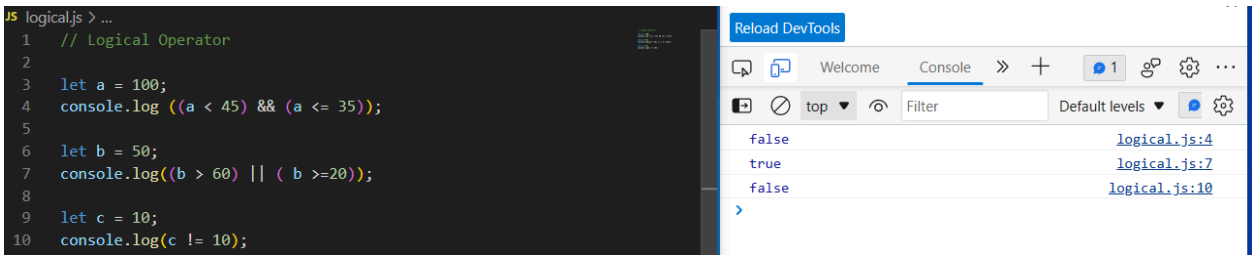
Logical operations used to determine the logic between variables or values.

- && → compare both the side of values are equal
- || → compare the values if one side of value are equal it returns true
- ! → not equal

Example:

```
let a = 100;
console.log((a < 45) && (a >= 35));   // AND Operator
let b = 50;
console.log(( b > 60) || (b >= 20));       // OR Operator
```

```
let c = 10;
console.log( 10 != 10);    // NOT Operator
```



0

E. Bitwise Operators:

Bit operators work on 32 bits numbers.Any numeric operand in the operation is converted into a 32 bit number (Convert the binary to decimal numbers).

- & → compare binary numbers if two bits have 1 then return 1.

Eg:

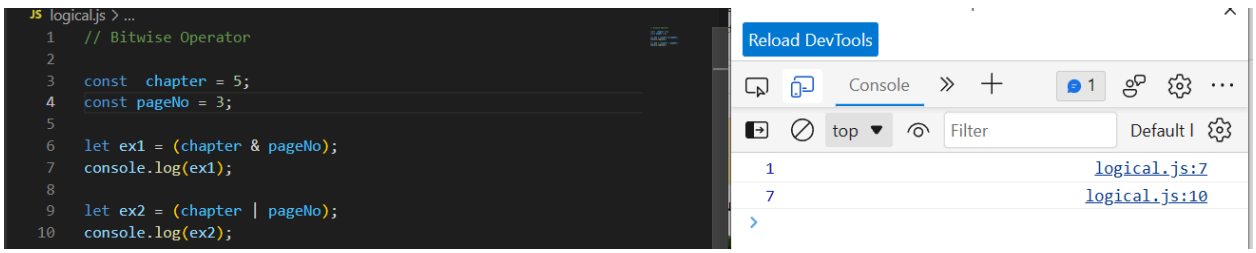
```
let x = 5;  —> 0101
let y = 3;  —> 0011
let z = (x & y)  —> 0101 & 0011    // Bitwise AND
console.log(z)  —>  0001
```

- | → compare binary numbers if any one bit has 1 then returns 1.

Eg: Let's take same input as shown above

```
let z = (x | y)  —> 0101 | 0011      // Bitwise OR
console.log(z)  —>  0111
```

Example:



F. Ternary Operator:

- The ternary conditional operator assigns a value to a variable based on a condition.

Example:

```
let mark = 85;
let result = (mark < 35) ? "Pass": "Fail";       // ternary operator
console.log(result);
```

31.05.2023

Control Statements

Conditional statements are used to perform different actions based on different conditions.

➢ If else:

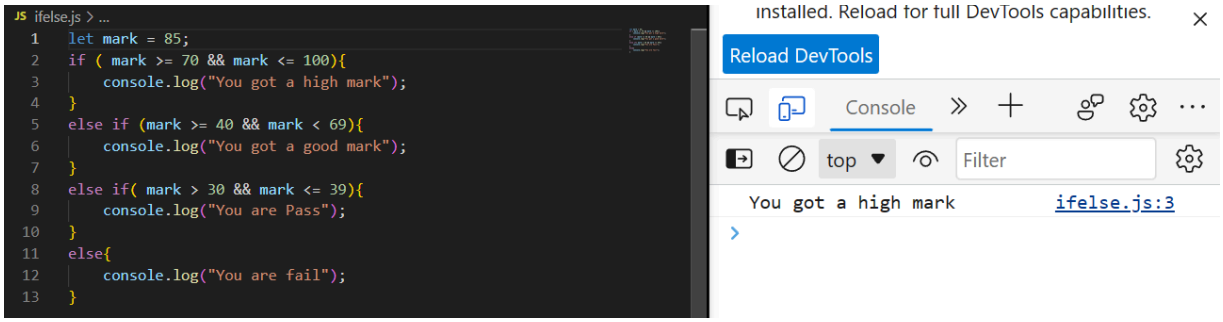
- Use if to specify a block of code to be executed, if a specified condition is true.
- Use else to specify a block of code to be executed, if the same condition is false.
- Use else if to specify a new condition to test, if the first condition is false

Example:

```
let mark = 85;

if( mark >= 70 && mark <= 100){
        console.log("You got a high mark")

}

else if( mark >= 40 && mark <= 69){
                console.log("You got a good mark")

}

else if( mark >= 30 && mark <= 39){
                console.log("You are pass")

}+
```

```
else{
                console.log(“You are fail”)

}+
```

- In the above example, the if statement checks the value of 85 is to be true in the following condition, if it is true then stop the execution.



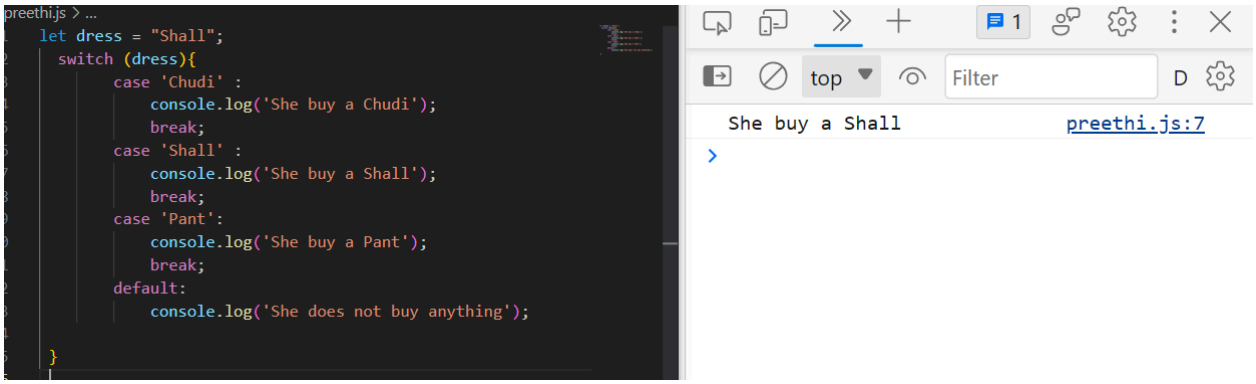
➢ Switch:

- The switch statement is used to perform different actions based on different conditions.
- Use the switch statement to select one of many code blocks to be executed.

Example:

```
let dress = "Shall";

switch (dress){
    case 'Chudi' :
        console.log('She buy a Chudi');
        break;
    case 'Shall' :
        console.log('She buy a Shall');
        break;
    case 'Pant':
        console.log('She buy a Pant');
        break;
    default:
        console.log('She does not buy anything');
}
```

- In the above example, the switch statement checks each case against the value of dress until the value is found. If nothing matches a default condition will be executed.



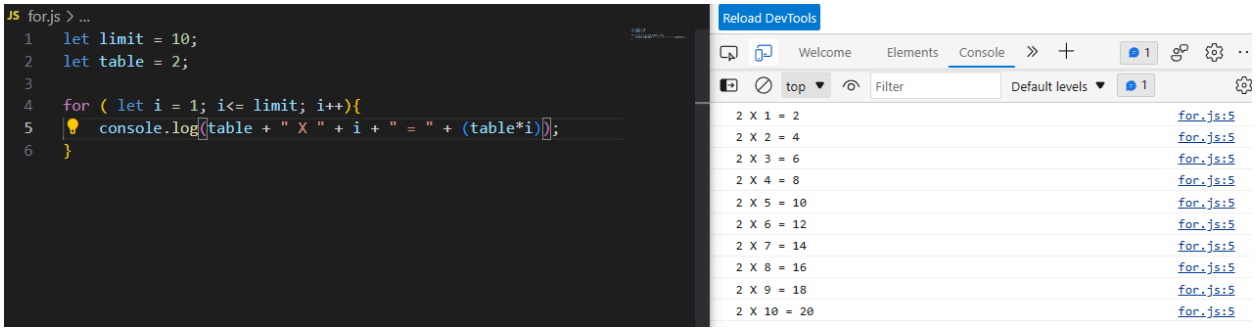
➢ For:

- The for loop is used to repeat a block of code for a specified condition is true**.**

Example:

```
// printing the multiplication table.
let limit = 10;
let table = 2;
for ( let i = 1; i<= limit; i++){
    console.log(table + " X " + i + " = " + (table*i));

}
```

- In the above example, we have to execute the multiplication table using the for loop which is used to repeat a code for a condition.



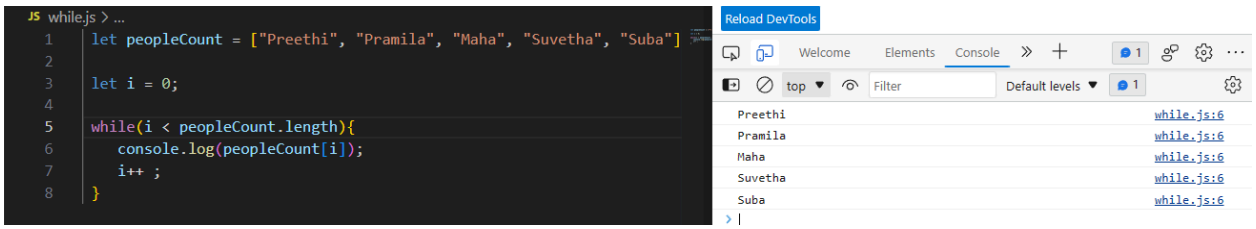
➢ While:

- The purpose of the while loop is to execute a statement repeatedly as long as an expression is true, otherwise it becomes false.

Example:

```
let peopleCount = ["Preethi", "Pramila", "Maha", "Suvetha", "Suba"];

let i = 0;
while(i < peopleCount.length){
 console.log(peopleCount[i]);
i++ ;
}
```

- In the above example, while loop is to execute an array repeatedly as a condition is true,then store the array values in i.



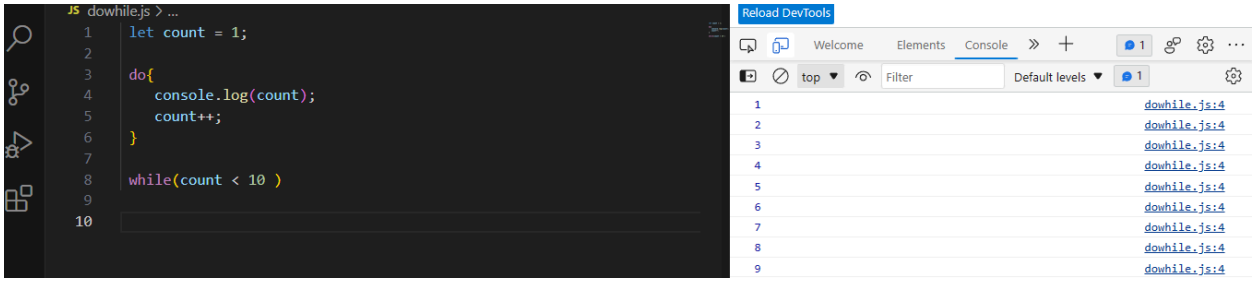
➢ do- while:

- The do-while is similar to the while loop except the condition checks , then executes a code as long as the condition is true.

Example:

```
 let count = 1;
      do{
      console.log(count);
      Count++;
}
while(count < 10 )
```

- In the above example, do-while loop will execute a code once before checking if the condition is true, then repeat the execute ends.



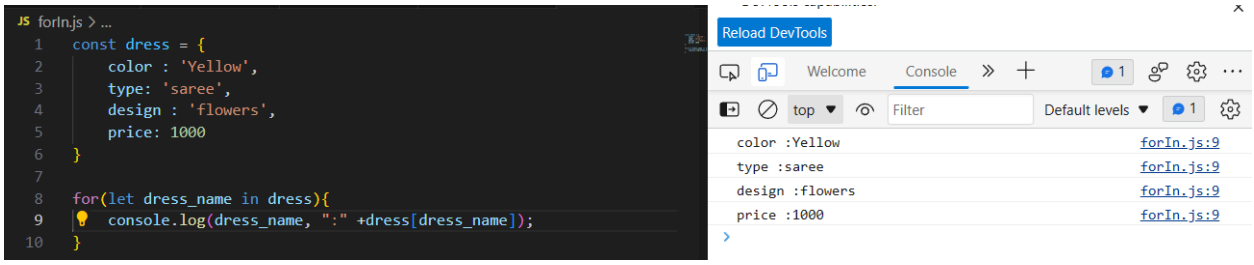
➢ For..In:

- The for in loop is used to loop through an object's properties.

Example:

```
const dress = {
        color : 'Yellow',
        type: 'saree',
        design : 'flowers',
       price: 1000
}
for(let dress_name in dress){
        console.log(dress_name, ":" +dress[dress_name]);
}
```

- In the above example, a for in loop is used to iterate over the dress_name properties in an object of dress.



➢ For…Of:

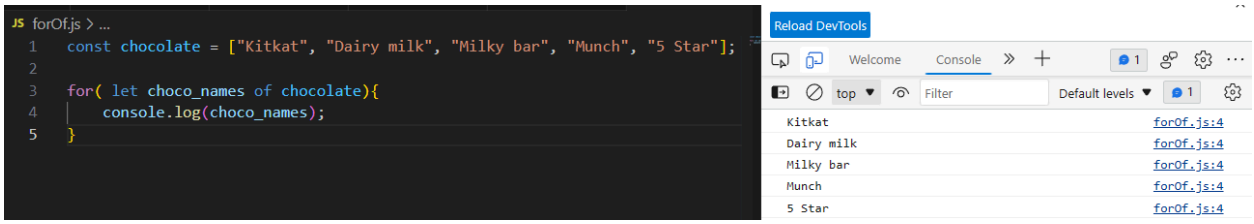
- The for of loop is used to loop through iterable like arrays, Strings, etc,.

Example:

```
const chocolate = ["Kitkat", "Dairy milk", "Milky bar", "Munch", "5 Star"];

for( let choco_names of chocolate){
```

```
    console.log(choco_names);
}
```

- In the above example, the for of loop is to iterate over the elements or items in an array of chocolate.



➢ Break:

- The break statement is used to terminate or jump out of the current loops as well as the switch statement.

Example:

```
let place = ["Chennai", "Cuddalore", "Madurai", "Ooty", "Pondycherry","Bangalore"];
let i = 0;
while(i < place.length){
        if (i === 4){
        break;
}
console.log(place[i]);
            i++;
 }
```

- In the above example, the break statement is used to execute the condition, if the index 4 is an array of place then jump out of a program.

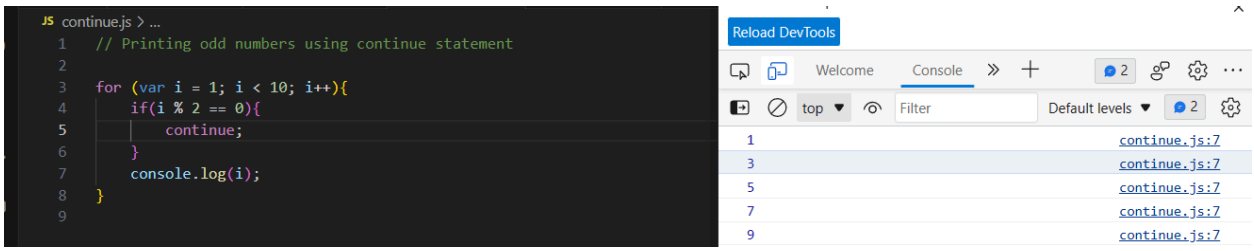
➢ Continue:

- The continue statement is used to skip the current iteration of the loop and jump to the next iteration.

Example:

```
// Printing odd numbers using continue statement
for (var i = 1; i < 10; i++){
        if(i % 2 == 0){
        continue;
}
         console.log(i);
}
```

- In the above example, we have to execute the odd numbers using the continue statement. If condition i modulus two is equal to 0 then continue between the odd numbers.



Data Types:

Primitive data types:

- String → " Apple, Mango" or ' Gova, Grapes'
- Number → ex: 23,4,100
- BigInt → ex: 15 digits and above
- Boolean → ex: true, false
- Undefined → value not declared
- Null → null value but it is object type
- Symbol → It represents a unique value
- Object → It is a method for encoding key-value pairs.

Strings

It is used for storing and manipulating the values.

➔ String methods.

➢ **Length()** – Find the length of the string.

Eg:

```
let text = "Preethi";
console.log(text.length);     // output 7.
```

➢ **slice()** - The slice() method is used to return a new array containing a portion of that array. It does not modify the original array rather returns a new array.
Syntax: slice(optional start parameter, optional end parameter)
Eg:

```
let fruits = ["Apple","Banana","Mango","Orange","Grapes"];
console.log(fruits.slice(2));     //  Output - ["Mango", "Orange", "Grapes"]
console.log(fruits.slice(1,-2)); //  Output - ["Banana", "Mango"]
```

➢ splice() – The splice() method is used to add or remove elements of an existing array and the return value will be the removed items from the array.
Syntax: splice(start, optional delete count, optional items to add)
Eg:

```
const months = ['Jan', 'March', 'April', 'June'];
months.splice(1, 0, 'Feb'); // Output - ['Jan', 'Feb', 'March', 'April', 'June']
months.splice(4, 1, 'May'); // Output - ['Jan', 'Feb', 'March', 'April', 'May']
months.splice(-3,2);     // Output - ['Jan', 'Feb', 'May']
console.log(months);
```

➢ **substring()** – It is similar to slice().The difference is that start and end values less than 0 are treated as 0 in substring().
Eg:

```
let str = "Van, Car, Bus";
console.log(str.substring(4, 8));  // output-  Car
```

➢ **Replace()** – This method replaces a specified value.
Eg:

```
let str = "Good Morning";
console.log(str.replace("Morning", "Evening");  // Output-  Good Evening
```

➢ **ReplaceAll()** – If replace all the values then use replaceAll() method.
Eg:

```
let name = "abcde";
Console.log(name.replaceAll("fhijkl")  // Output - fhijkl
```

➢ **toUpperCase()** – converted to uppercase.
Eg:

```
let text1 = "Hello World!";
console.log(text1.toUpperCase()); // HELLO WORLD
```

➢ **toLowerCase()** – converted to uppercase.

Eg:

```
let txt = "HELLO";
console.log(text1.toLowerCase());  // hello
```

➢ **Concat()** – joins two or more strings.

Eg:

```
let txt1 = "Welcome";
let txt2 = "Guys";
console.log(txt1.concat(text2)) // Welcome Guys
```

➢ **trim()** – removes whitespace from both sides of a string.

Eg:

```
let text = "     Welcome     ";
let result = text.trim();
console.log(result);   // Welcome
```

➢ **trimStart()** – removes whitespace only from the start of a string.

Eg:

```
let greet = "   Thankyou";
let result = greet.trim();
console.log(greet.trimStrat())  // Thankyou
```

➢ **trimEnd()** – removes whitespace only from the end of a string.

Eg:

```
let greet = "Sorry    ";
let greet= greet.trim();
console.log(greet.trimStrat())  // Sorry
```

➢ **padStart()** – pads a string from the start.

Eg:

```
let a = 4
console.log(a.padStart(3, “0”))  //   004
```

➢ **padEnd()** – pads a string from the end.

Eg:

```
let b = 5
```

```
console.log(a.padStart(4, “xx”))  //   4xx
```

➢ **charAt()** – returns the character at a specified index in a string.

Eg:

```
let car = “Volve”
console.log(car.charAt(3)) //   v
```

➢ **charCodeAt()** – returns the unicode of the character at a specified index in a string.

Eg:

```
let car = “Volve”
console.log(car.charCodeAt(0)) //   86
```

➢ **Split()** – A string can be converted to an array.

Eg:

```
const str = 'Hello World';
const chars = str.split(' '); //Output - [‘Hello’,’World’]
const chars = str.split(''); //Output - ['H',’e’,’l’,’l’,’o’, ‘’,’W’,’o’,’r’,’l’,’d’]
const chars = str.split('r'); //Output - ['Hello Wo', 'ld']
const chars = str.split(); // Output - ['Hello World']
console.log(chars);
```

➔ Search Methods:

There are eight search methods in string, such as:

- indexOf() → It returns the index of the first occurrence of a specified text in a string.
- eturns the index of the last occurrence of a specified text in a String.
  → This methods searches backwards (from the end to the beginning), meaning: if the second parameter is 15, the search starts at position 15, and searches to the beginning of the string.
- search() → It searches the specified text in a string and returns its position of the match.

- match() → This method returns an array containing the results of matching a string against a string. If a regular expression does not include the g modifier (global search..(i.e., → /ain/g)), match() will return only the first match in the string.

- matchAll() → This method returns an iterator containing the results of matching a string against a string. It also has global search and case sensitivity..

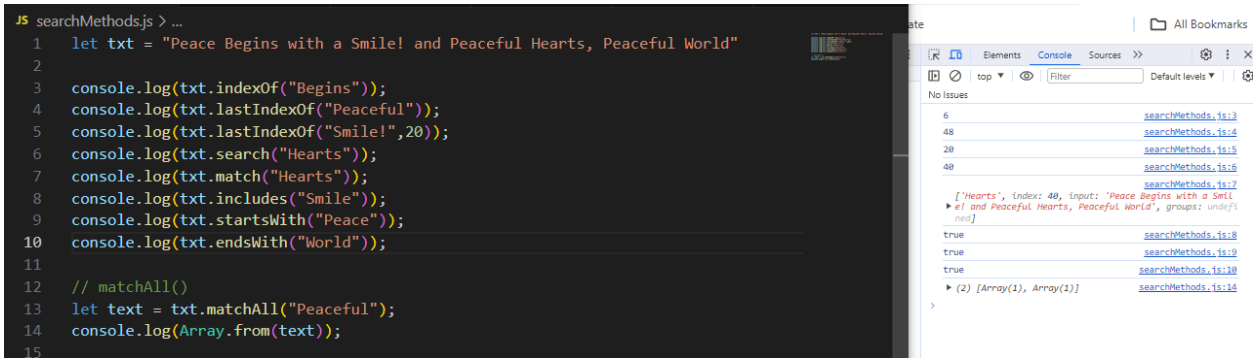
- includes() → This method returns true if a string contains a specified value. Otherwise it returns false.

- startsWith() → It returns true if a string begins with a specified value. Otherwise it returns false.

- endsWith() → It returns true if a string begins with a specified value. Otherwise it returns false.

Example:

```
let txt = "Peace Begins with a Smile! and Peaceful Hearts, Peaceful World"

console.log(txt.indexOf("Begins"));
console.log(txt.lastIndexOf("Peaceful"));
console.log(txt.lastIndexOf("Smile!",20));
console.log(txt.search("Hearts"));
console.log(txt.match("Hearts"));
console.log(txt.includes("Smile"));
console.log(txt.startsWith("Peace"));
console.log(txt.endsWith("World"));
// matchAll()
let text = txt.matchAll("Peaceful");
console.log(Array.from(text));
```



```
searchMethods.js > ...
1  let txt = "Peace Begins with a Smile! and Peaceful Hearts, Peaceful World"
2
3  console.log(txt.indexOf("Begins"));
4  console.log(txt.lastIndexOf("Peaceful"));
5  console.log(txt.lastIndexOf("Smile!",20));
6  console.log(txt.search("Hearts"));
7  console.log(txt.match("Hearts"));
8  console.log(txt.includes("Smile"));
9  console.log(txt.startsWith("Peace"));
10 console.log(txt.endsWith("World"));
11
12 // matchAll()
13 let text = txt.matchAll("Peaceful");
14 console.log(Array.from(text));
15
```

```
All Bookmarks
Elements  Console  Sources
top  Filter  Default levels
No Issues
6                                   searchMethods.js:3
48                                  searchMethods.js:4
20                                  searchMethods.js:5
40                                  searchMethods.js:6
                                    searchMethods.js:7
  ['Hearts', index: 40, input: 'Peace Begins with a Smil
▸ e! and Peaceful Hearts, Peaceful World', groups: undefi
  ned]
true                                searchMethods.js:8
true                                searchMethods.js:9
true                                searchMethods.js:10
▸ (2) [Array(1), Array(1)]          searchMethods.js:14
```

Number

JavaScript has only one type of number. Numbers can be written with or without decimals.

➔ Number Methods:

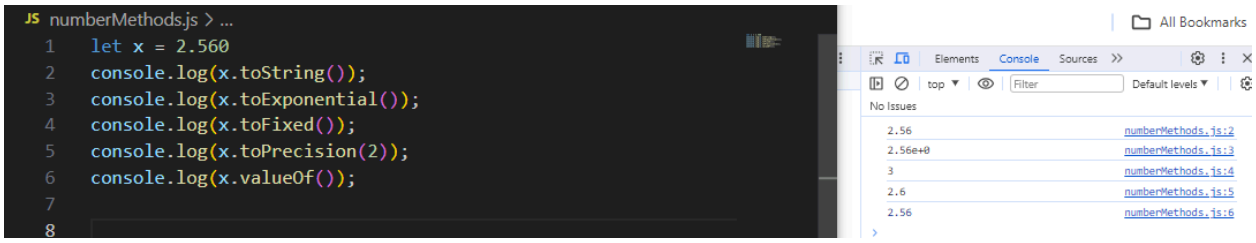
These number methods can be used on all JavaScript numbers:

- toString() → Returns a number as a string
- toExponential() → Returns a number written in exponential notation
- toFixed() → Returns a number written with a number of decimals. And it is perfect for working with money.

- toPrecision() → Returns a number written with a specified length
- ValueOf() → Returns a number as a number

Example:

```
let x = 2.560
console.log(x.toString());
console.log(x.toExponential());
console.log(x.toFixed());
console.log(x.toPrecision(2));
console.log(x.valueOf());
```



Arrays

An array is a special variable, which can hold more than one value.

Eg: const cars = ["Saab", "Volvo", "BMW"];

Array Methods:

There are eight search methods in string, such as

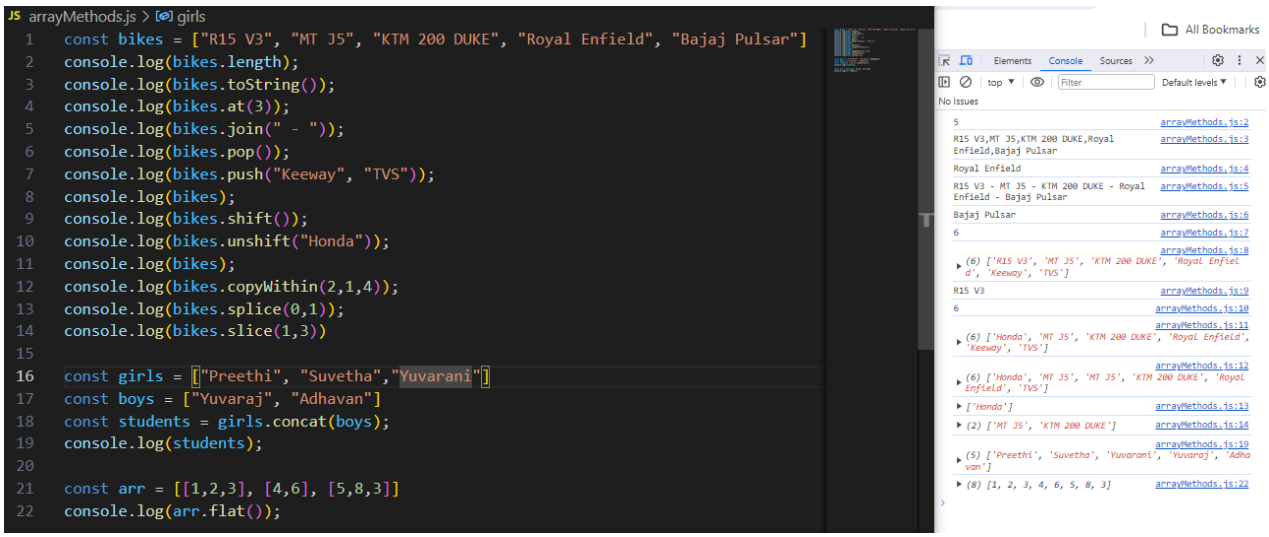
- length →It returns the length (size) of an array.
- toString() →It converts an array to a string of (comma separated) array values.
- at() → It returns an indexed element from an array
- join() →It joins all array elements into a string
- pop() → It removes the last element from an array
- push() → It adds a new element to an array (at the end).
- shift() → It removes the first array element and "shifts" all other elements to a lower index.
- unshift() → It adds a new element to an array (at the beginning), and "unshifts" older elements

- copyWithin() → It copies array elements to another position in an array
- concat() → It creates a new array by merging (concatenating) existing arrays.
- flat() → It creates a new array with sub-array elements concatenated to a specified depth and used to convert a multidimensional array into a one-dimensional array.
- splice() → It removes elements without leaving "holes" in the array.
- slice() → It slices out a piece of an array into a new array

Example:

const bikes = ["R15 V3", "MT J5", "KTM 200 DUKE", "Royal Enfield", "Bajaj Pulsar"]

```
console.log(bikes.length);
console.log(bikes.toString());
console.log(bikes.at(3));
console.log(bikes.join(" - "));
console.log(bikes.pop());
console.log(bikes.push("Keeway", "TVS"));
console.log(bikes);
console.log(bikes.shift());
console.log(bikes.unshift("Honda"));
console.log(bikes);
console.log(bikes.copyWithin(2,1,4));
console.log(bikes.splice(0,1));
console.log(bikes.slice(1,3));

const girls = ["Preethi", "Suvetha","Yuvarani"]
const boys = ["Yuvaraj", "Adhavan"]
const students = girls.concat(boys);
console.log(students);

const arr = [[1,2,3], [4,6], [5,8,3]]
console.log(arr.flat());
```



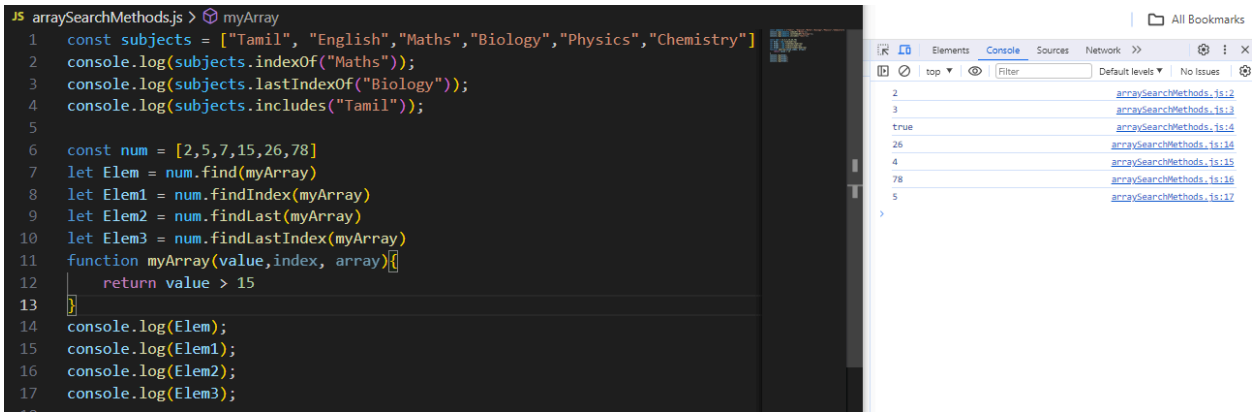
Array Search Methods:

- indexOf() → It searches an array for an element value and returns its position and it returns -1 if the item is not found.If the item is present more than once, it returns the position of the first occurrence.

- lastIndexOf → It is the same as Array.indexOf(), but returns the position of the last occurrence of the specified element.
- includes() → It allows us to check if an element is present in an array or not.
- find() → It returns the value of the first array element that passes a test function.
- findIndex() → It returns the index of the first array element that passes a test function.
- findLast() → This method that will start from the end of an array and return the value of the first element that satisfies a condition.
- findLastIndex() → finds the index of the last element that satisfies a condition.

Example:

```
const subjects = ["Tamil", "English","Maths","Biology","Physics","Chemistry"]
console.log(subjects.indexOf("Maths"));
console.log(subjects.lastIndexOf("Biology"));
console.log(subjects.includes("Tamil"));

const num = [2,5,7,15,26,78]
let Elem = num.find(myArray)
let Elem1 = num.findIndex(myArray)
let Elem2 = num.findLast(myArray)
let Elem3 = num.findLastIndex(myArray)
function myArray(value,index, array){
   return value > 15
}
console.log(Elem);
console.log(Elem1);
console.log(Elem2);
console.log(Elem3);
```



Object

- In JavaScript, an object is a standalone entity, with properties and type.
- A JavaScript object is the collection of named values.
- A common practice to declare objects with the const keyword.

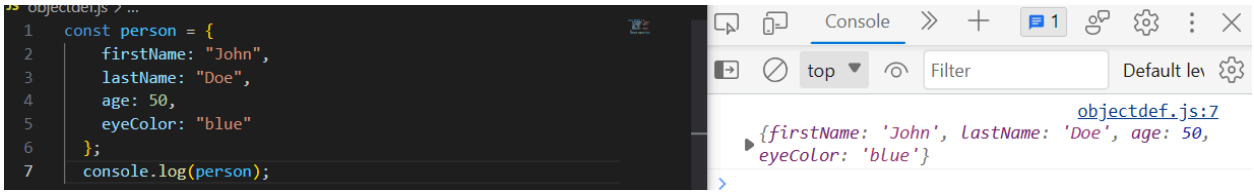
There are different ways to create new objects:

- Create a single object, using an object literal.
- Create a single object, with the keyword new.
- Define an object constructor, and then create objects of the constructed type.
- Create an object using Object.create().

➔ using an object literal:

- This is the easiest way to create a JavaScript Object.
- Using an object literal, you both define and create an object in one statement.
- An object literal is a list of name:value pairs inside curly braces {}.
- ***Example:***

```
const person = {
    firstName: "John",
    lastName: "Doe",
    age: 50,
   eyeColor: "blue"
};
console.log(person);
```

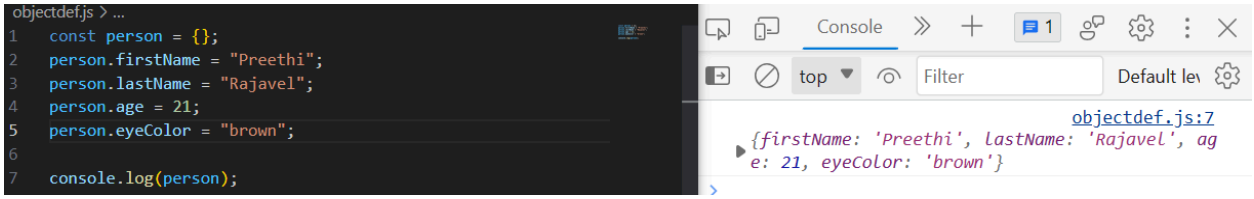


➔ creates an empty JavaScript object:

- **Example:**

```
const person = {};
```

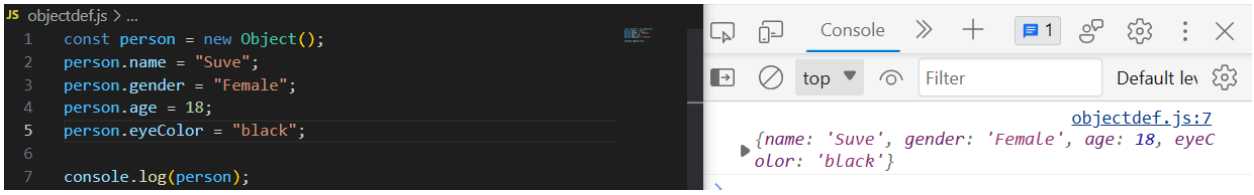
```
person.firstName = "Preethi";
person.lastName = "Rajavel";
person.age = 2;
person.eyeColor = "brown";
console.log(person);
```



➔ create a new JavaScript object using new Object().

- **Example:**

```
const person = new Object{};

person.name = "Suve";
person.gender = "Female";
person.age = 8;
person.eyeColor = "black";

console.log(person);
```



Object Properties:

- Properties are the values associated with a JavaScript object.
- A JavaScript object is a collection of unordered properties.
- Properties can usually be changed, added, and deleted, but some are read only.

Some common solutions to display JavaScript objects are:

- Displaying the Object Properties by name
- Displaying the Object Properties in a Loop
- Displaying the Object using Object.values()

- Displaying the Object using JSON.stringify()

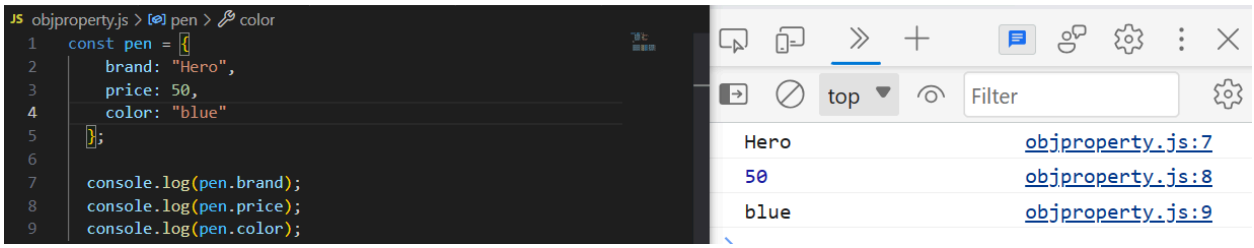
The syntax for accessing the property of an object is:

objectName.property // person.age

objectName["*property*"] // person["age"]

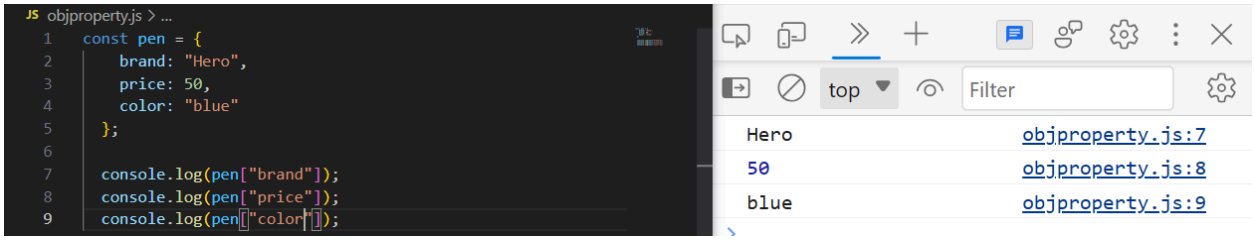
- **Example-I:**

```
const pen= {
    brand:"Hero",
    price:50,
    color:”blue”
};
console.log(pen.brand);
```



- **Example-II:**

```
const person = {
     brand:"Hero",
    price:50,
    color:”blue”
};
console.log(pen[“brand”]);
```



➢ **JavaScript for...in Loop:**

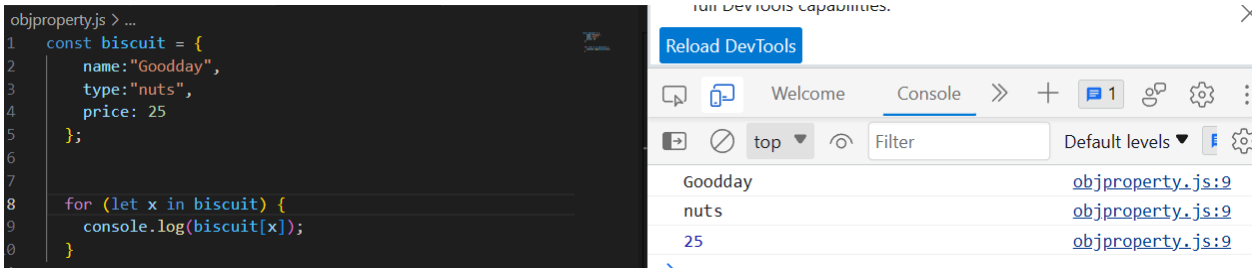
The JavaScript for...in statement loops through the properties of an object.

Syntax:

```
for (let variable in object) {
// code to be executed
}
```

- **Example:**

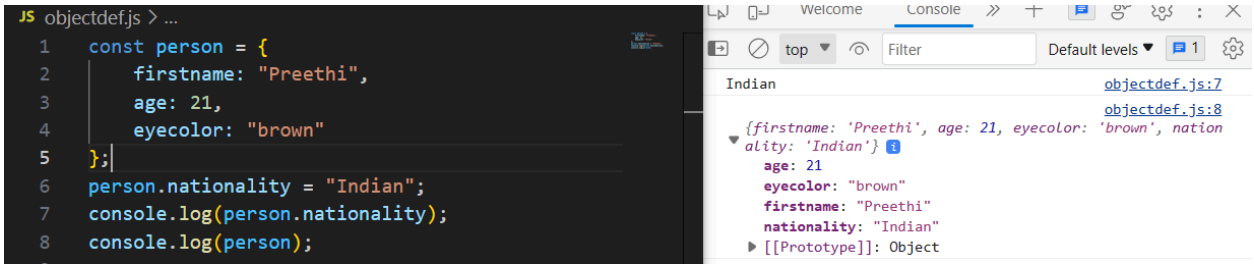
```
const biscuit = {
      name:"Goodday",
      type:"nuts",
      price: 25
};

 for (let x in biscuits) {
  console.log(biscuits[x];
}
```



➔ **Adding New Properties:**

- You can add new properties to an existing object by simply giving it a value.
- Example:

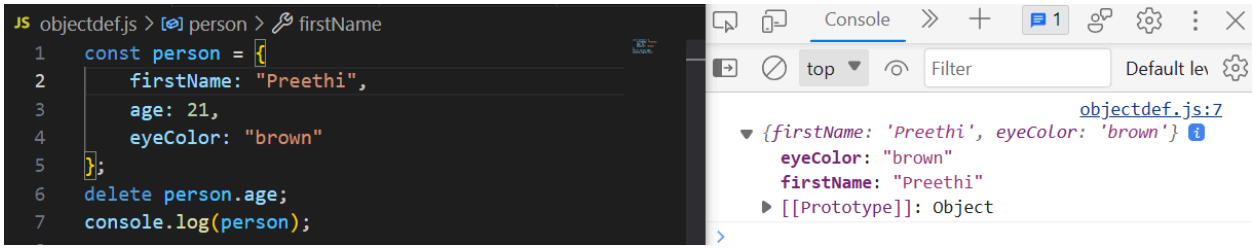
```
const person = {
        firstname: "Preethi",
       age: 50,
        eyecolor: "blue"
};
person.nationality = "English";
console.log(person.nationality);
console.log(person);
```



➔ **Deleting Properties:**

- The delete keyword deletes a property from an object:
- Example:

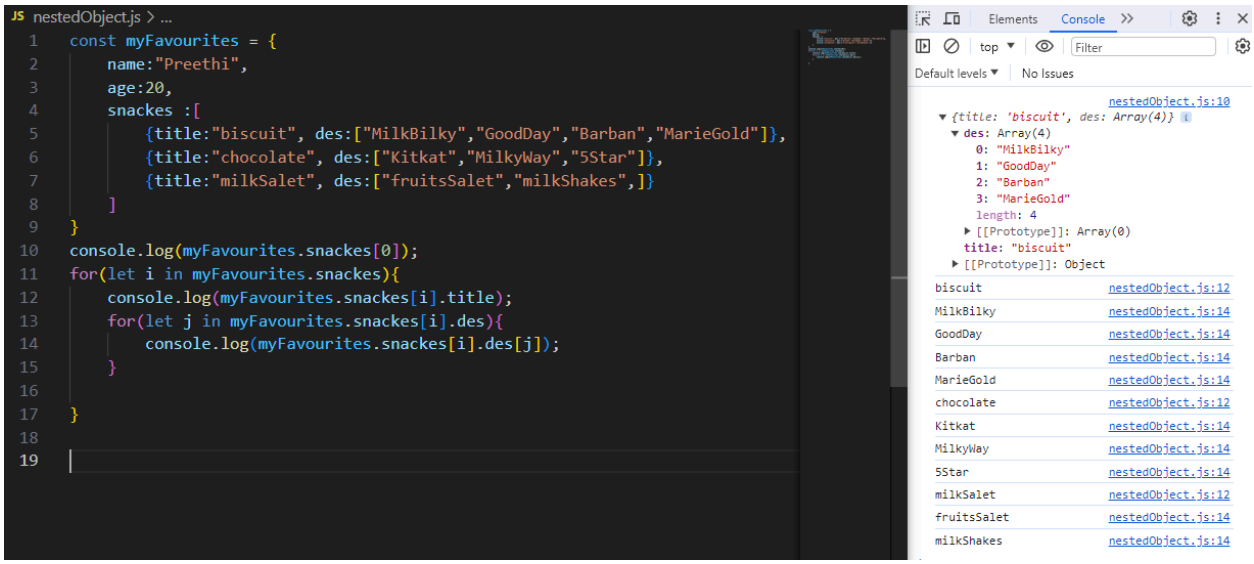
```
const person = {
        firstName: "Preethi",
        age: 21,
        eyeColor: "brown"
};
delete person.age;
console.log(person);
```



→ **Objects of Arrays:**

- Values in objects can be arrays, and values in arrays can be objects
- Example:

```
const myFavourites = {
        name:"Preethi",
        age:20,
        snackes :[
                {
                    title:"biscuit",
                  des:["MilkBilky","GoodDay","Barban","MarieGold"]
                },
                {
                    title:"chocolate",
                    des:["Kitkat","MilkyWay","5Star"]
                },
                {
                    title:"milkSalet",
                    des:["fruitsSalet","milkShakes",]
                }
        ]
}
console.log(myFavourites.snackes[0]);
        for(let i in myFavourites.snackes){
                console.log(myFavourites.snackes[i].title);
                    for(let j in myFavourites.snackes[i].des){
                            console.log(myFavourites.snackes[i].des[j]);
                    }

        }
```

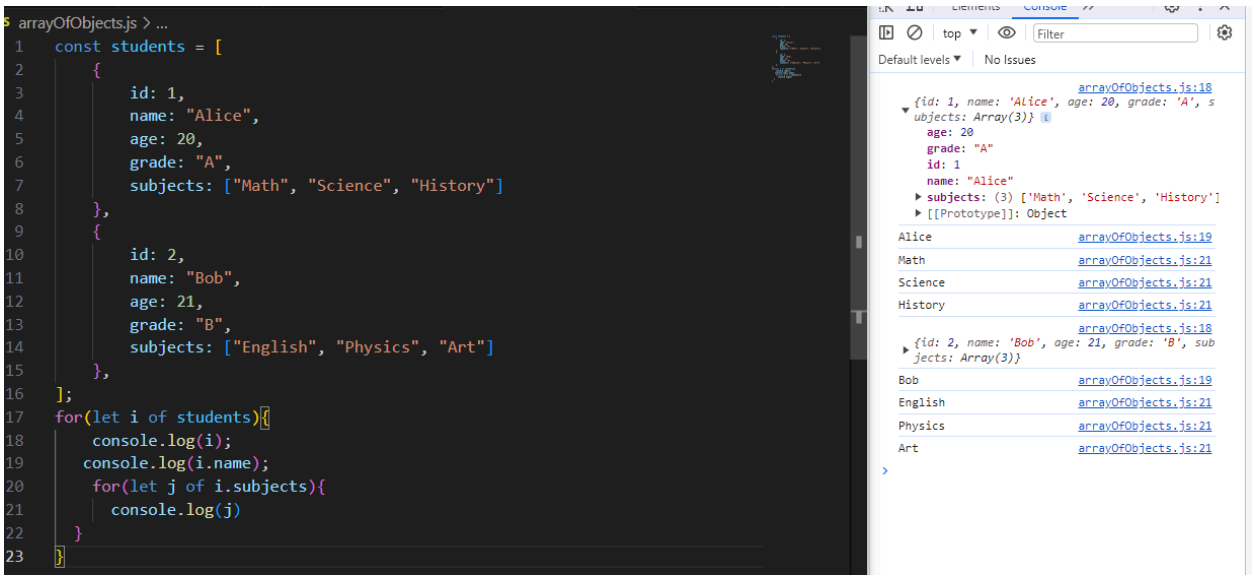


→ **Arrays of Objects:**

- Values in Array can be objects, and values in objects can be array.
- Example:

```
const students = [
    {
        id: 1,
        name: "Alice",
        age: 20,
        grade: "A",
        subjects: ["Math", "Science", "History"]
    },
    {
        id: 2,
        name: "Bob",
        age: 21,
        grade: "B",
        subjects: ["English", "Physics", "Art"]
    },
];
for(let i of students){
   console.log(i);
   console.log(i.name);
        for(let j of i.subjects){
                console.log(j)
```

```
        }
    }
```



```
arrayOfObjects.js > ...
const students = [
    {
        id: 1,
        name: "Alice",
        age: 20,
        grade: "A",
        subjects: ["Math", "Science", "History"]
    },
    {
        id: 2,
        name: "Bob",
        age: 21,
        grade: "B",
        subjects: ["English", "Physics", "Art"]
    },
];
for(let i of students){
    console.log(i);
   console.log(i.name);
    for(let j of i.subjects){
      console.log(j)
  }
}
```

```
top
Filter
Default levels
No Issues
arrayOfObjects.js:18
{id: 1, name: 'Alice', age: 20, grade: 'A', subjects: Array(3)}
  age: 20
  grade: "A"
  id: 1
  name: "Alice"
  subjects: (3) ['Math', 'Science', 'History']
  [[Prototype]]: Object
Alice                    arrayOfObjects.js:19
Math                     arrayOfObjects.js:21
Science                  arrayOfObjects.js:21
History                  arrayOfObjects.js:21
arrayOfObjects.js:18
{id: 2, name: 'Bob', age: 21, grade: 'B', subjects: Array(3)}
Bob                      arrayOfObjects.js:19
English                  arrayOfObjects.js:21
Physics                  arrayOfObjects.js:21
Art                      arrayOfObjects.js:21
```

Object Methods:

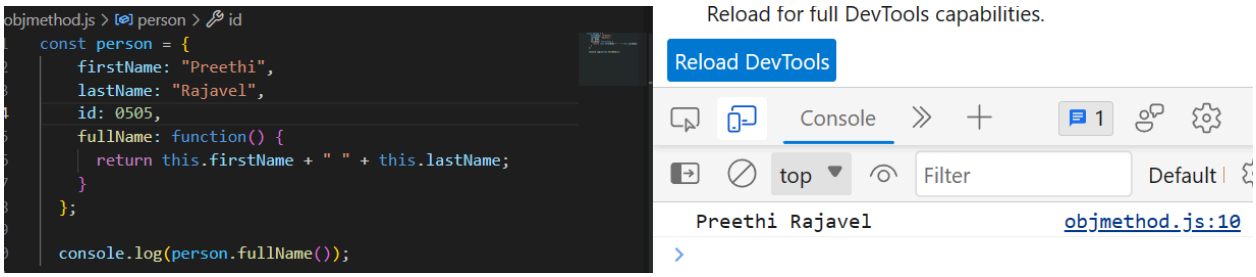
- JavaScript methods are actions that can be performed on objects.
- A JavaScript method is a property containing a function definition.
- Methods are functions stored as object properties.
- You access an object method with the following syntax:

objectName.methodName()

- **Example:**

```
const person = {
    firstName: "Preethi",
    lastName: "Rajavel",
    id: 0505,
    fullName: function() {
       return     this.firstName + " " + this.lastName;
     }
    };
```

```
console.log(person.fullName());
```

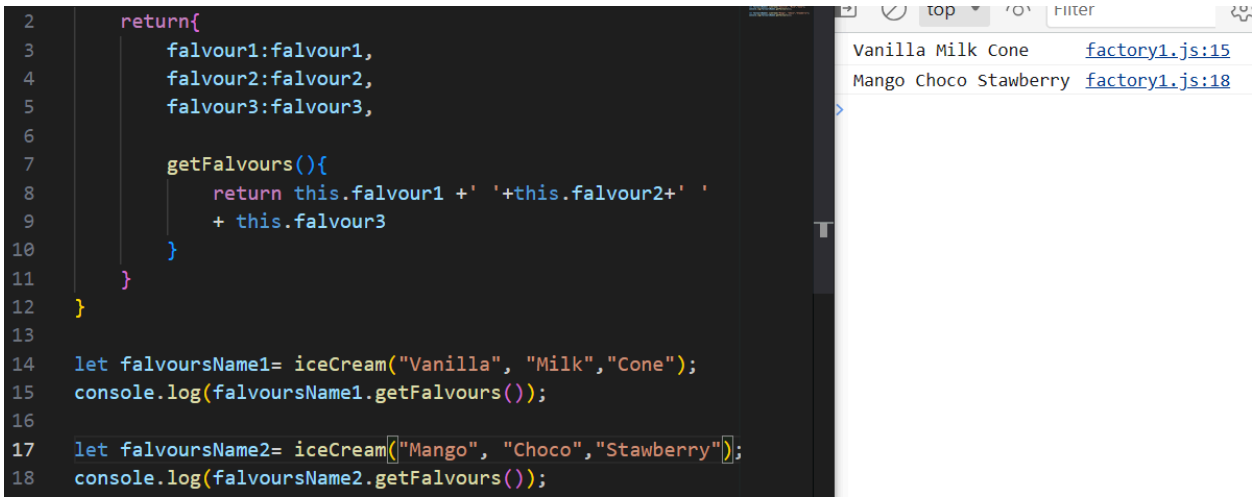


Factory function in object:

- A factory function can be defined as a function that creates an object and returns it.
- A factory function is a function that returns a new object.
- To avoid copying the same code, you can define a function that creates the object to access a value.
- When a function creates and returns a new object, it is called a factory function. The iceCream() is a factory function because it returns a new flavourName object.
- **Example:**

```
function iceCream(flavour 1,flavour 2,flavour 3){
 return{
          falvour1:falvour1,
         falvour2:falvour2,
         falvour3:falvour3,

  getFalvours(){
      return this.falvour1 +' '+this.falvour2+' '
       + this.falvour3
   }
  }
}

let falvoursName1= iceCream("Vanilla", "Milk","Cone");
console.log(falvoursName1.getFalvours());
```

```
let falvoursName2= iceCream("Mango", "Choco","Stawberry");
console.log(falvoursName2.getFalvours());
```

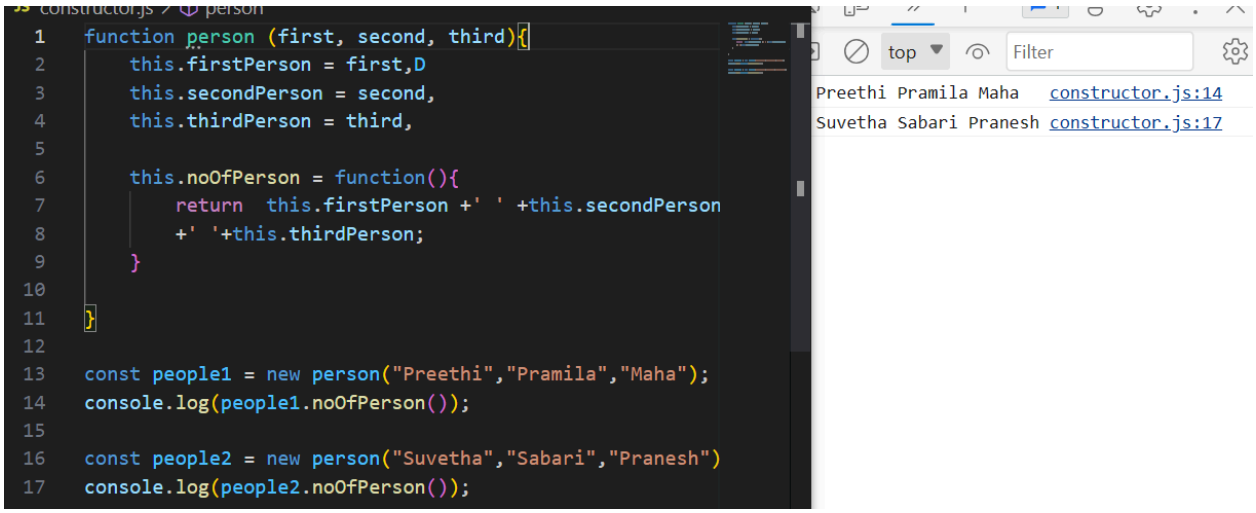


Constructor function in object:

- A constructor is a special function that creates and initializes an object instance of a class.
- In JavaScript, a constructor gets called when an object is created using the new keyword.
- The purpose of a constructor is to create a new object and set values for any existing object properties.
- In a constructor function this does not have a value. It is a substitute for the new object. The value of this will become the new object when a new object is created.
- A constructor function should be called only with the new operator.
- **Example:**

```
function person (first, second, third){
    this.firstPerson = first,
    this.secondPerson = second,
    this.thirdPerson = third,

    this.noOfPerson = function(){
        return  this.firstPerson +' ' +this.secondPerson
```

```
            +' '+this.thirdPerson;
        }

    }

    const people1 = new person("Preethi","Pramila","Maha");
    console.log(people1.noOfPerson());

    const people2 = new person("Suvetha","Sabari","Pranesh");
    console.log(people2.noOfPerson());
```



Object Prototypes:

All JavaScript objects inherit properties and methods from a prototype.

Prototype Inheritance:

All JavaScript objects inherit properties and methods from a prototype:

- Date objects inherit from Date.prototype
- Array objects inherit from Array.prototype
- Person objects inherit from Person.prototype
- The Object.prototype is on the top of the prototype inheritance chain:

Date objects, Array objects, and Person objects inherit from Object.prototype

Using the prototype Property:

- ➔ It allows you to add new properties to object constructors
- ➔ It also allows you to add new methods to objects constructors

Example:

```
function person (first, last, age,gender){
        this.firstName = first;
        this.lastName = last;
       this.personAge = age;
       this.personGender = gender
}
// adding a new method to the constructor object.
person.prototype.name = function(){
        return this.firstName + " " + this.lastName;
};
// adding new value to constructor object
//person.prototype.degree = "BE"

const personDetails = new person("Ram", "Sai", 23, "Male");
//console.log(personDetails.degree);

console.log(personDetails.name());
```

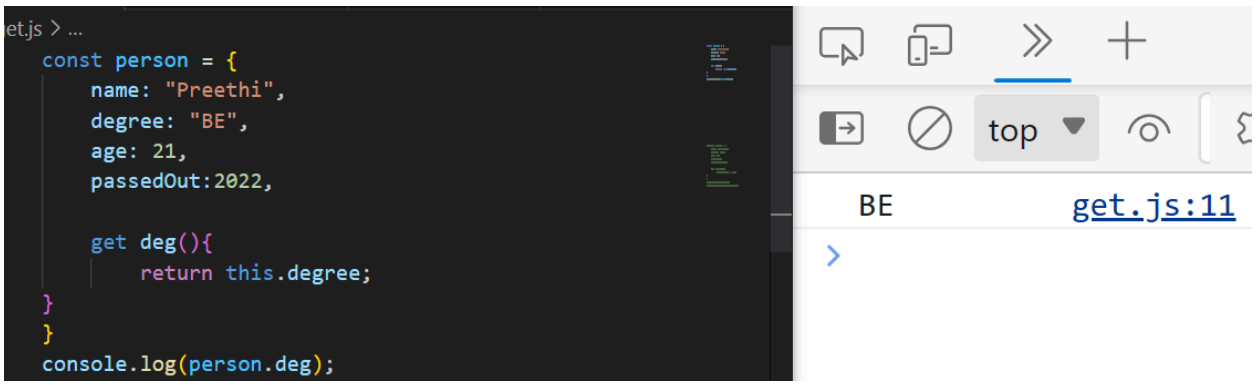
JavaScript Accessors:

Getters and setters allow you to define Object Accessors (Computed Properties).

➔ Get Method:

- The Javascript getter methods are used to access the properties of an object.
- **Example:**

```
const person = {
     name: "Preethi",
    degree: "BE",
     age: 21,
    passedOut:2022,

   get deg(){
       return this.degree;
   }
}
console.log(person.deg);
```

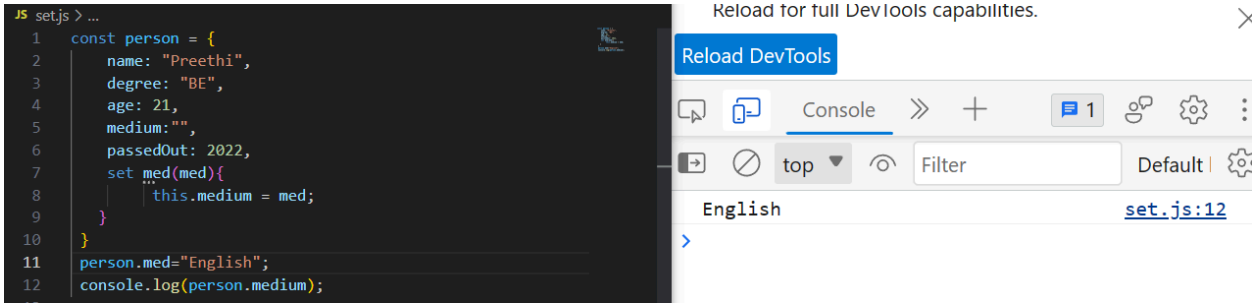


➔ Set Method:

- The Javascript setter methods are used to change the values of an object.
- **Example:**

```
const person = {
   name: "Preethi",
   degree: "BE",
   age: 21,
   medium:"",
```

```
  passedOut: 2022
  set med(med){
      this.medium = med;
  }
}
person.med="English";
console.log(person.medium);
```



Sets:

- A JavaScript Set is a collection of unique values.
- Each value can only occur once in a Set.
- A Set can hold any value of any data type.

Set Methods:

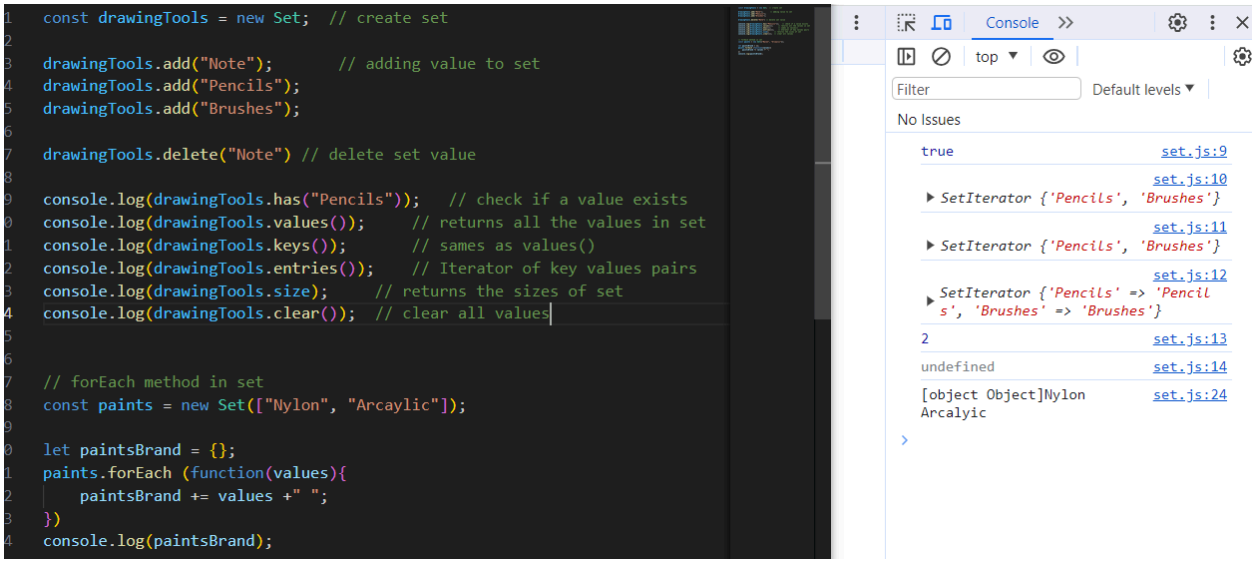
- ➔ new Set() → Creates a new Set
- ➔ add() → Adds a new element to the Set
- ➔ delete() → Removes an element from a Set
- ➔ has() → Returns true if a value exists
- ➔ clear() → Removes all elements from a Set
- ➔ forEach() → Invokes a callback for each element
- ➔ values() → Returns an Iterator with all the values in a Set
- ➔ keys() → Same as values()
- ➔ entries() → Returns an Iterator with the [value,value] pairs from a Set

Property:

- ➢ size → Returns the number elements in a Set

Example:

```
const drawingTools = new Set;  // create set

drawingTools.add("Note");       // adding value to set
drawingTools.add("Pencils");
drawingTools.add("Brushes");

drawingTools.delete("Note") // delete set value

console.log(drawingTools.has("Pencils"));   // check if a value exists
console.log(drawingTools.values());     // returns all the values in set
console.log(drawingTools.keys());       // sames as values()
console.log(drawingTools.entries());    // Iterator of key values pairs
console.log(drawingTools.size);     // returns the sizes of set
console.log(drawingTools.clear());  // clear all values

// forEach method in set
const paints = new Set(["Nylon", "Arcaylic"]);

let paintsBrand = {};
paints.forEach (function(values){
        paintsBrand += values +" ";
})
console.log(paintsBrand);
```



Map:

- A Map holds key-value pairs where the keys can be any datatype.
- A Map remembers the original insertion order of the keys.
- A Map has a property that represents the size of the map.

Map Methods:

- ➔ new Map() → Creates a new Map object
- ➔ set() → Sets the value for a key in a Map
- ➔ get() → Gets the value for a key in a Map
- ➔ clear() → Removes all the elements from a Map
- ➔ delete() → Removes a Map element specified by a key
- ➔ has() → Returns true if a key exists in a Map
- ➔ forEach() → Invokes a callback for each key/value pair in a Map
- ➔ entries() → Returns an iterator object with the [key, value] pairs in a Map
- ➔ keys() → Returns an iterator object with the keys in a Map
- ➔ values() → Returns an iterator object of the values in a Map

Property

- ➔ size → Returns the number of Map elements

Example:

```
const paints = new Map([
            ["brand","Nylon"],
           ["price", 250]
]);                  // create a map
paints.set("qty", 3)    // add the values in map
console.log(paints);
console.log(paints.get('price'));  //   access the value from map
console.log(paints.has('qty'));  // check if a key value pair is exist
console.log(paints.keys());     // returns keys of map
console.log(paints.values());   // returns values of map
console.log(paints.size);       //returns size of map
console.log(paints.entries());              // returns iterator
console.log(paints.delete("brand"));  // delete a value in map
console.log(paints.clear());    // clear all the values in map

// forEach() in map
```

```
const colors = new Map([
        ["Yellow", 3],
        ["Sky blue", 5]
]);
let likeColors = "";
colors.forEach (function(value, key){
        likeColors += key +":" +value +" "
})
console.log(likeColors);
```