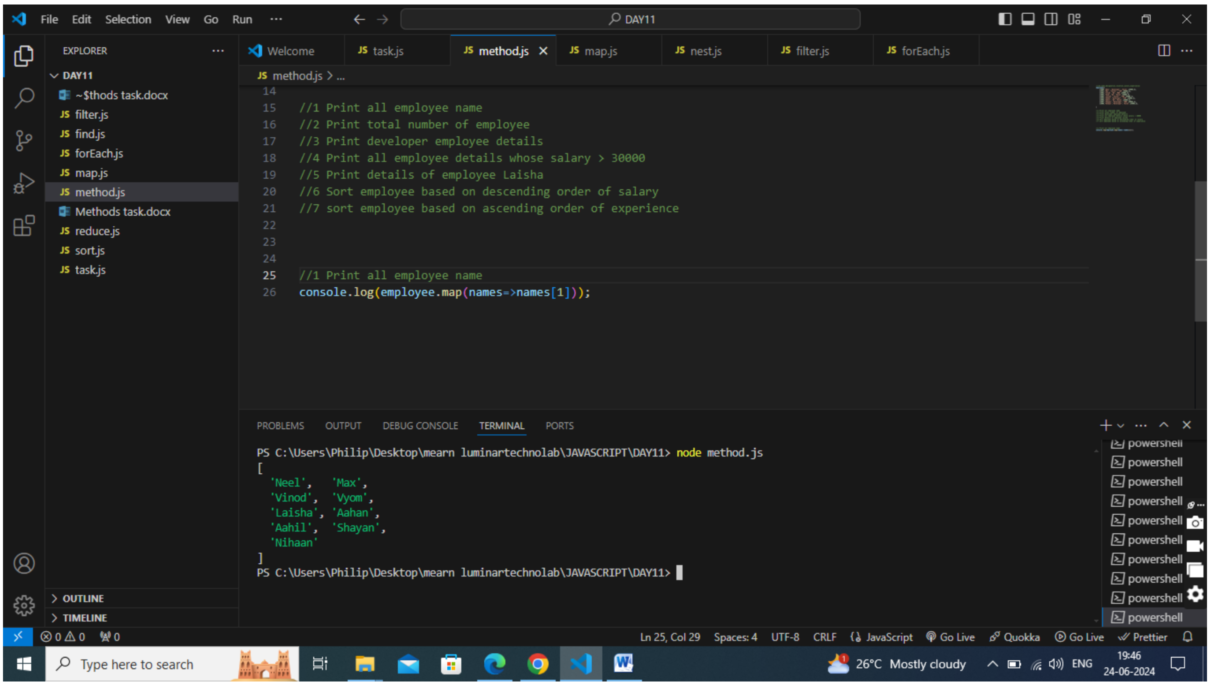
TASK1

```
//[id,name,designation,location,salary,experience]

employee=[

    [1000,'Neel','Developer','Kochi',25000,3],

    [1001,'Max','Tester','TVM',20000,2],

    [1002,'Vinod','QA','KNR',35000,4],

    [1003,'Vyom','QA','Kochi',45000,5],

    [1004,'Laisha','Tester','TVM',55000,7],

    [1005,'Aahan','Developer','TVM',15000,1],

    [1006,'Aahil','QA','Kochi',25000,3],

    [1007,'Shayan','Developer','KNR',30000,3],

    [1000,'Nihaan','Developer','Kochi',25000,3],


]
```

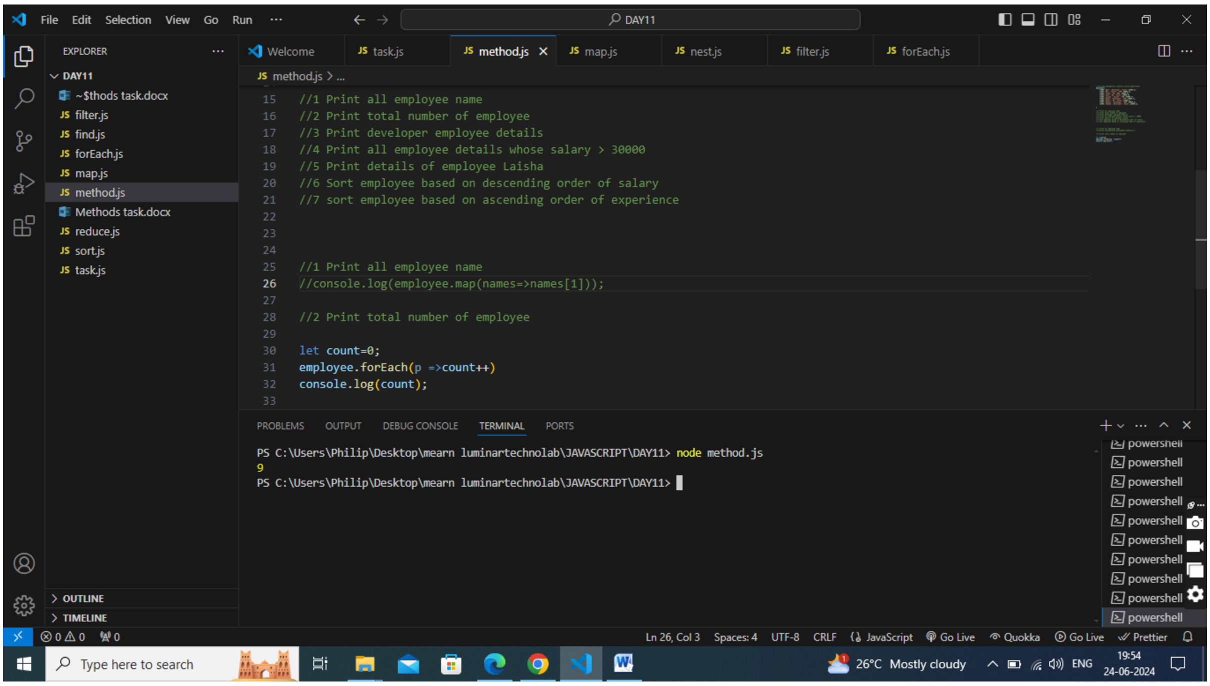
//1 Print all employee name



```
//1 Print all employee name
//2 Print total number of employee
//3 Print developer employee details
//4 Print all employee details whose salary > 30000
//5 Print details of employee Laisha
//6 Sort employee based on descending order of salary
//7 sort employee based on ascending order of experience



//1 Print all employee name
console.log(employee.map(names=>names[1]));
```

```
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11> node method.js
[
  'Neel',   'Max',
  'Vinod',  'Vyom',
  'Laisha', 'Aahan',
  'Aahil',  'Shayan',
  'Nihaan'
]
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11>
```

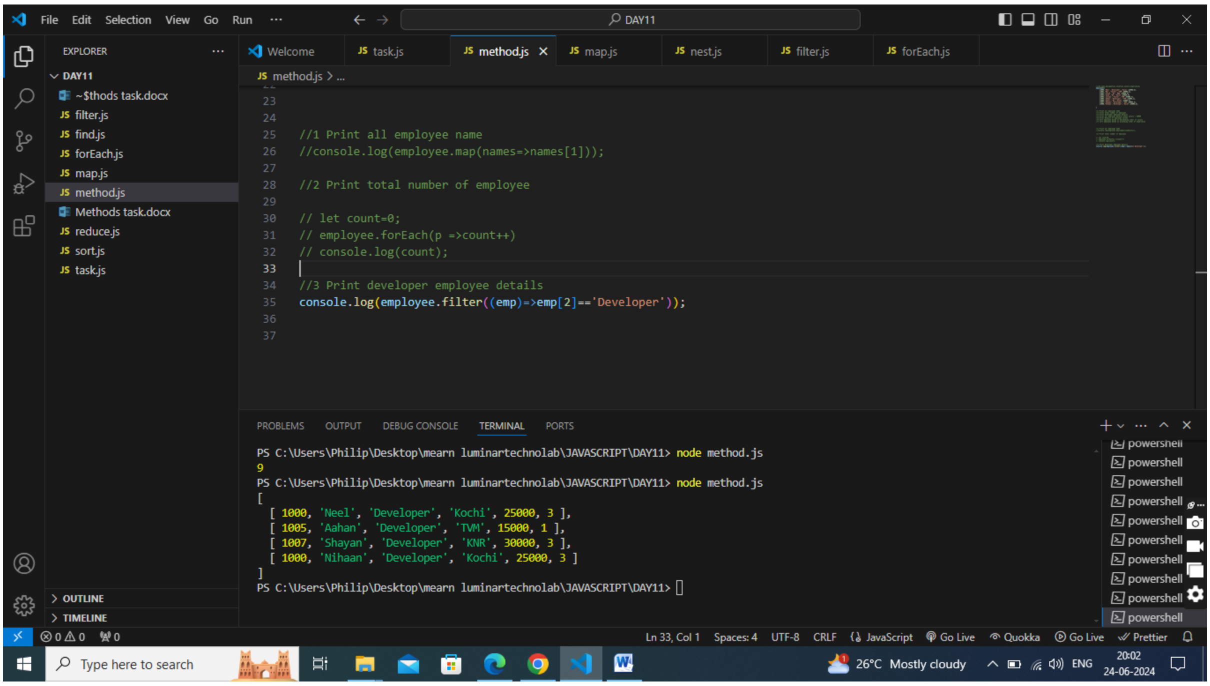
//2 Print total number of employee



```
//1 Print all employee name
//2 Print total number of employee
//3 Print developer employee details
//4 Print all employee employee details whose salary > 30000
//5 Print details of employee Laisha
//6 Sort employee based on descending order of salary
//7 sort employee based on ascending order of experience



//1 Print all employee name
//console.log(employee.map(names=>names[1]));

//2 Print total number of employee

let count=0;
employee.forEach(p =>count++)
console.log(count);

```

```
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11> node method.js
9
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11> 
```

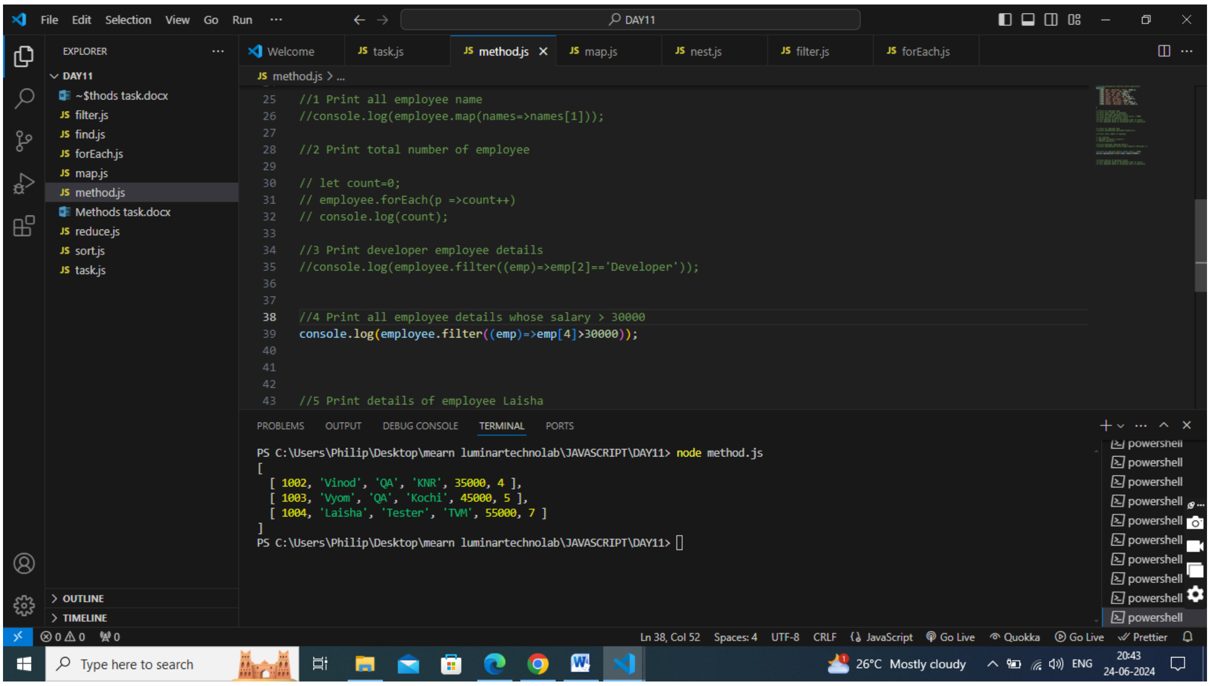
//3 Print developer employee details



```


//1 Print all employee name
//console.log(employee.map(names=>names[1]));

//2 Print total number of employee

// let count=0;
// employee.forEach(p =>count++)
// console.log(count);

//3 Print developer employee details
console.log(employee.filter((emp)=>emp[2]=='Developer'));


```

```
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11> node method.js
9
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11> node method.js
[
  [ 1000, 'Neel', 'Developer', 'Kochi', 25000, 3 ],
  [ 1005, 'Aahan', 'Developer', 'TVM', 15000, 1 ],
  [ 1007, 'Shayan', 'Developer', 'KNR', 30000, 3 ],
  [ 1000, 'Nihaan', 'Developer', 'Kochi', 25000, 3 ]
]
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11> 
```

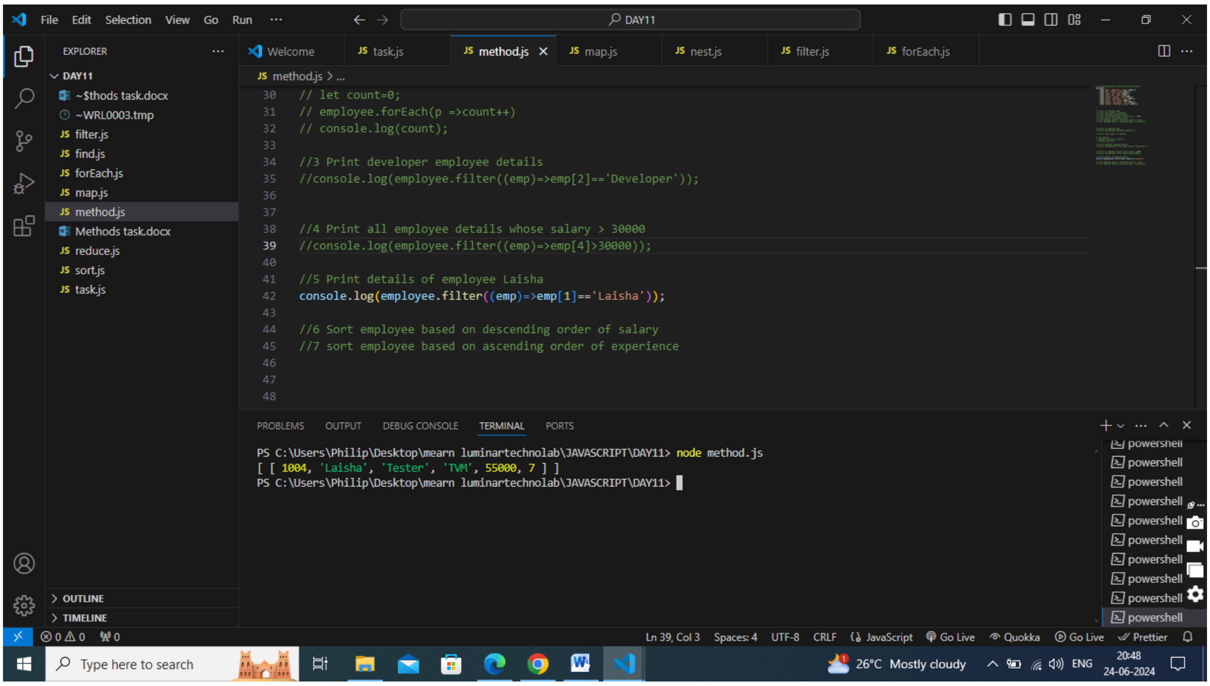
//4 Print all employee details whose salary > 30000



```javascript
//1 Print all employee name
//console.log(employee.map(names=>names[1]));

//2 Print total number of employee

// let count=0;
// employee.forEach(p =>count++)
// console.log(count);

//3 Print developer employee details
//console.log(employee.filter((emp)=>emp[2]=='Developer'));


//4 Print all employee details whose salary > 30000
console.log(employee.filter((emp)=>emp[4]>30000));



//5 Print details of employee Laisha
```

```
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11> node method.js
[
  [ 1002, 'Vinod', 'QA', 'KNR', 35000, 4 ],
  [ 1003, 'Vyom', 'QA', 'Kochi', 45000, 5 ],
  [ 1004, 'Laisha', 'Tester', 'TVM', 55000, 7 ]
]
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11>
```

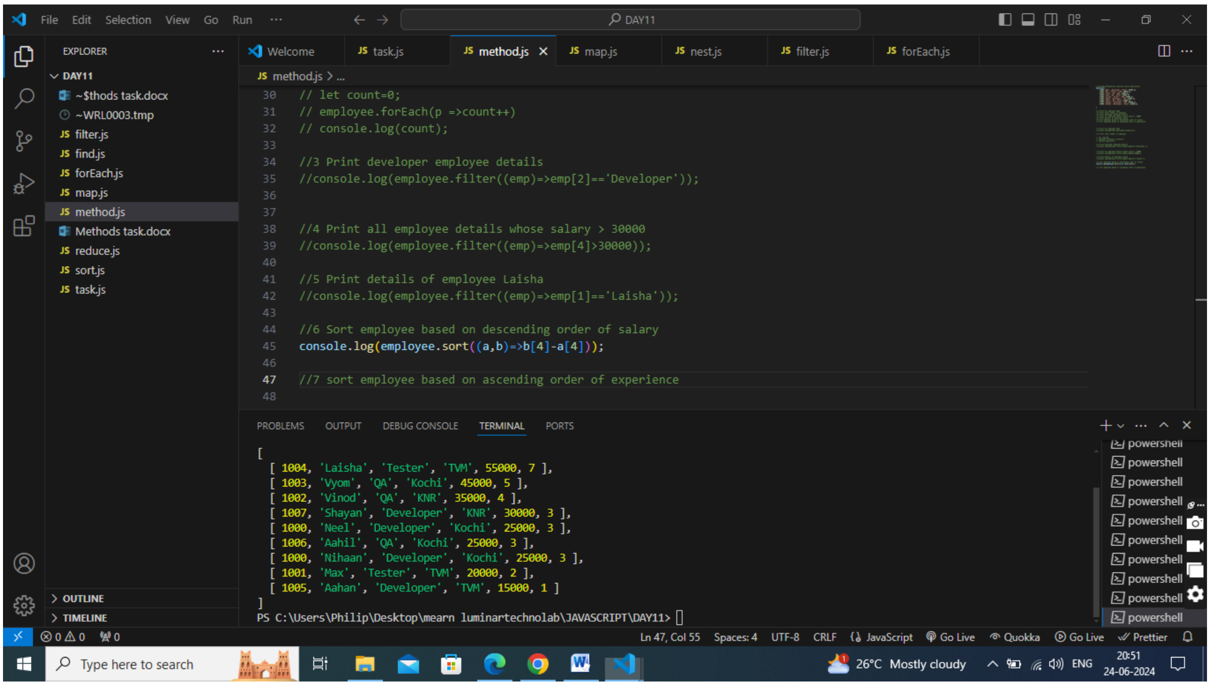
//5 Print details of employee Laisha



```javascript
// let count=0;
// employee.forEach(p =>count++)
// console.log(count);

//3 Print developer employee details
//console.log(employee.filter((emp)=>emp[2]=='Developer'));


//4 Print all employee details whose salary > 30000
//console.log(employee.filter((emp)=>emp[4]>30000));

//5 Print details of employee Laisha
console.log(employee.filter((emp)=>emp[1]=='Laisha'));

//6 Sort employee based on descending order of salary
//7 sort employee based on ascending order of experience
```

```
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11> node method.js
[ [ 1004, 'Laisha', 'Tester', 'TVM', 55000, 7 ] ]
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11>
```

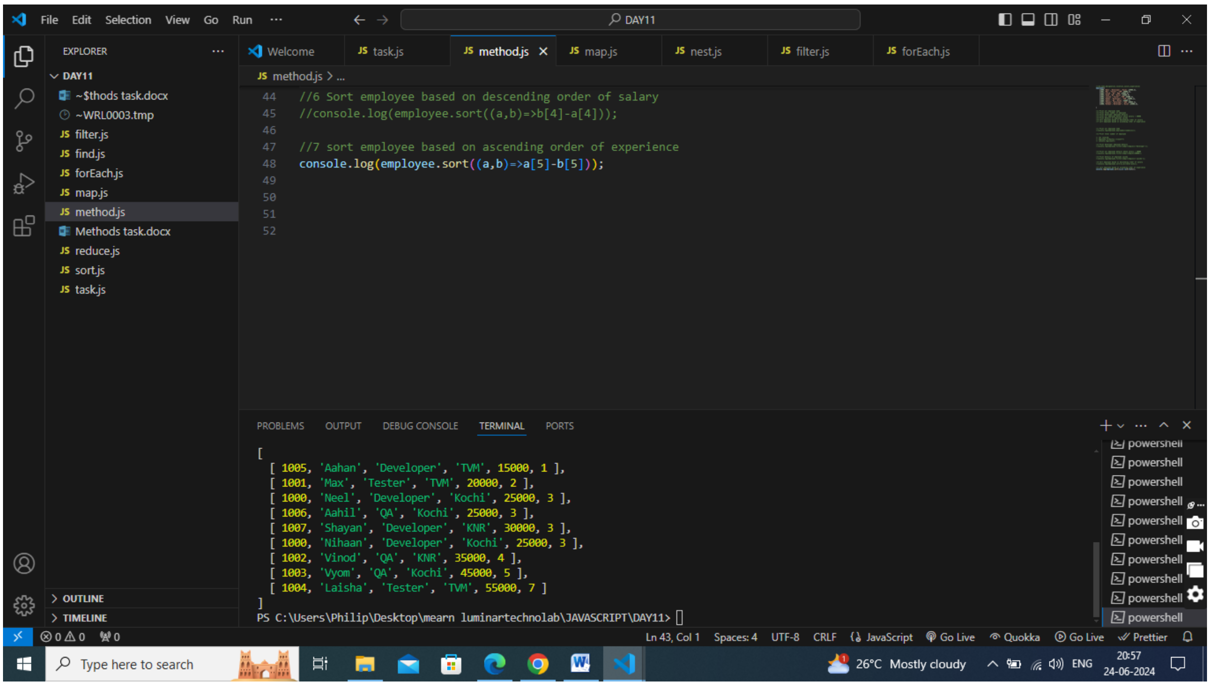
//6 Sort employee based on descending order of salary



```
// let count=0;
// employee.forEach(p =>count++)
// console.log(count);

//3 Print developer employee details
//console.log(employee.filter((emp)=>emp[2]=='Developer'));


//4 Print all employee details whose salary > 30000
//console.log(employee.filter((emp)=>emp[4]>30000));

//5 Print details of employee Laisha
//console.log(employee.filter((emp)=>emp[1]=='Laisha'));

//6 Sort employee based on descending order of salary
console.log(employee.sort((a,b)=>b[4]-a[4]));

//7 sort employee based on ascending order of experience

```

```
[
  [ 1004, 'Laisha', 'Tester', 'TVM', 55000, 7 ],
  [ 1003, 'Vyom', 'QA', 'Kochi', 45000, 5 ],
  [ 1002, 'Vinod', 'QA', 'KNR', 35000, 4 ],
  [ 1007, 'Shayan', 'Developer', 'KNR', 30000, 3 ],
  [ 1000, 'Neel', 'Developer', 'Kochi', 25000, 3 ],
  [ 1006, 'Aahil', 'QA', 'Kochi', 25000, 3 ],
  [ 1000, 'Nihaan', 'Developer', 'Kochi', 25000, 3 ],
  [ 1001, 'Max', 'Tester', 'TVM', 20000, 2 ],
  [ 1005, 'Aahan', 'Developer', 'TVM', 15000, 1 ]
]
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11>
```

//7 sort employee based on ascending order of experience



```
//6 Sort employee based on descending order of salary
//console.log(employee.sort((a,b)=>b[4]-a[4]));

//7 sort employee based on ascending order of experience
console.log(employee.sort((a,b)=>a[5]-b[5]));
```

```
[
  [ 1005, 'Aahan', 'Developer', 'TVM', 15000, 1 ],
  [ 1001, 'Max', 'Tester', 'TVM', 20000, 2 ],
  [ 1000, 'Neel', 'Developer', 'Kochi', 25000, 3 ],
  [ 1006, 'Aahil', 'QA', 'Kochi', 25000, 3 ],
  [ 1007, 'Shayan', 'Developer', 'KNR', 30000, 3 ],
  [ 1000, 'Nihaan', 'Developer', 'Kochi', 25000, 3 ],
  [ 1002, 'Vinod', 'QA', 'KNR', 35000, 4 ],
  [ 1003, 'Vyom', 'QA', 'Kochi', 45000, 5 ],
  [ 1004, 'Laisha', 'Tester', 'TVM', 55000, 7 ]
]
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11>
```

TASK2

1. What is the purpose of the Array.prototype.filter() method in JavaScript?

The Array.prototype.filter() method in JavaScript is used to create a new array with all elements that pass a test implemented by the provided function. The filter() method does not execute the function for empty elements. The filter() method does not change the original array.

Characteristics:

Non-mutating: The filter() method does not change the original array. Instead, it returns a new array.

Callback function: The method accepts a callback function that is executed on each element of the array.

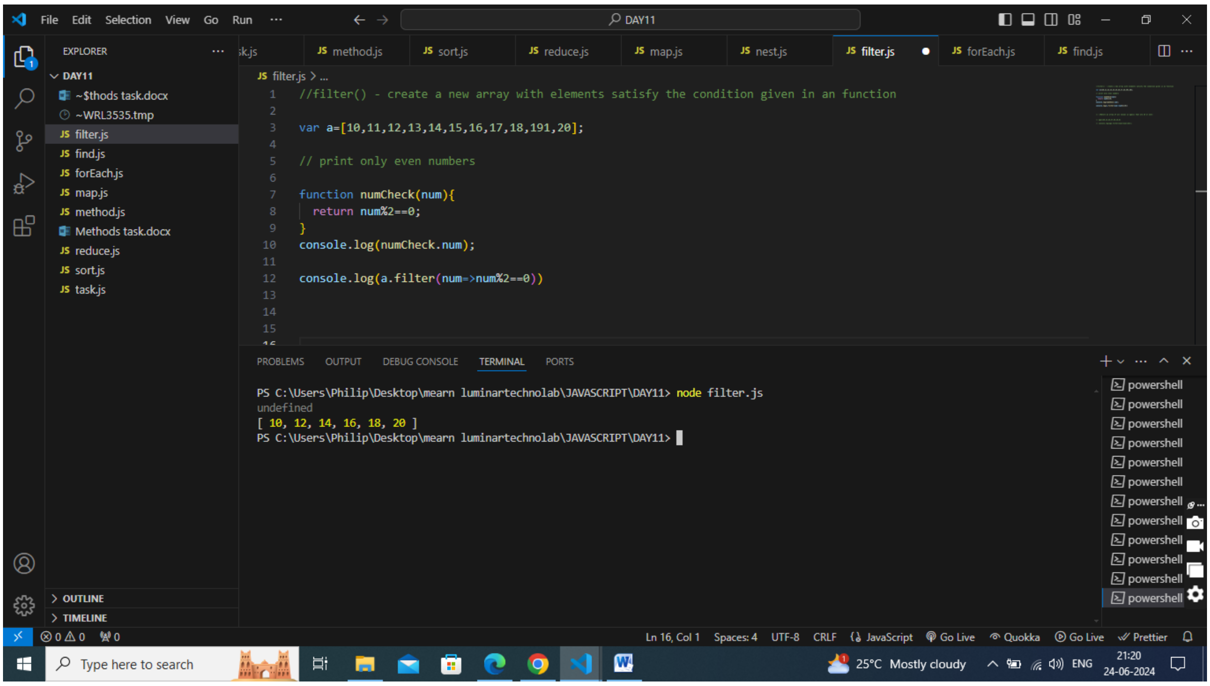
Boolean return: The callback function must return a boolean value (true or false). Only elements for which the callback returns true are included in the new array.

2. How does the filter() method work? Can you explain the basic idea behind its functionality?

The filter() method works by iterating over each element of an array and applying a specified callback function to determine whether each element should be included in a new array. If the callback function returns true for an element, that element is included in the new array; if it returns false, the element is excluded.

- Initialization: The method takes a callback function as an argument. This callback function contains the condition that each element of the array must meet to be included in the new array.
- Iteration: The filter() method iterates over each element in the array. For each element, it calls the callback function, passing the current element, its index, and the entire array as arguments.
- Condition Check: Inside the callback function, a condition is checked for each element. The callback function returns true if the element meets the condition, or false otherwise.
- Building the New Array: If the callback function returns true for an element, that element is added to a new array. If it returns false, the element is ignored.
- Returning the Result: After all elements have been processed, the filter() method returns the new array containing only the elements that passed the condition.

3. Can you demonstrate how to use the filter() method to create a new array of even numbers from an existing array of integers?



```
//filter() - create a new array with elements satisfy the condition given in an function

var a=[10,11,12,13,14,15,16,17,18,191,20];

// print only even numbers

function numCheck(num){
  return num%2==0;
}
console.log(numCheck.num);

console.log(a.filter(num=>num%2==0))
```

```
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11> node filter.js
undefined
[ 10, 12, 14, 16, 18, 20 ]
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11> 
```

4. How does the filter() method differ from the find() method in terms of functionality and returned values?

filter() Method

- ➢ Purpose: The filter() method is used to create a new array containing all elements from the original array that satisfy a specified condition.
- ➢ Returned Value: A new array containing all elements that pass the test implemented by the provided callback function. If no elements pass the test, it returns an empty array.

find() Method

- ➢ Purpose: The find() method is used to find the first element in the array that satisfies the provided testing function.
- ➢ Returned Value: The value of the first element that satisfies the condition. If no elements satisfy the condition, it returns undefined

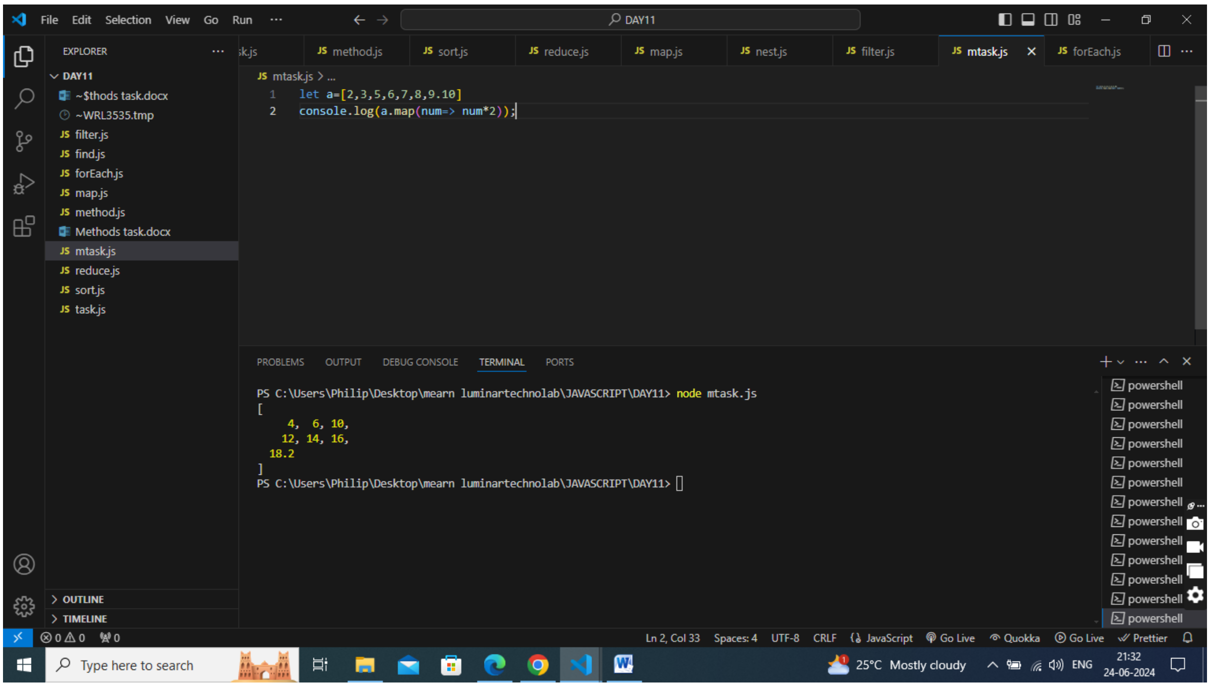
5. What is the purpose of the Array.prototype.map() method in JavaScript?

The Array.prototype.map() method in JavaScript is used to create a new array populated with the results of calling a provided function on every element in the calling array. This method is particularly useful for transforming the elements of an array without modifying the original array.

Key Characteristics

- ➢ Transformation: The map() method applies a provided callback function to each element in the array and constructs a new array from the returned values.
- ➢ Non-mutating: It does not modify the original array but instead returns a new array.
- ➢ Same Length: The new array will always have the same length as the original array.

6. Can you provide an example of using the map() method to double each element in an array of numbers?



```
let a=[2,3,5,6,7,8,9.10]
console.log(a.map(num=> num*2));
```

```
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11> node mtask.js
[
     4,  6, 10,
    12, 14, 16,
  18.2
]
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11>
```

//7 Can you explain the difference between the map() method and the forEach() method?

- ➢ Return Value: map(): Returns a new array containing the results of applying the callback function to each element.
- ➢ forEach(): Returns undefined.

Use Case:

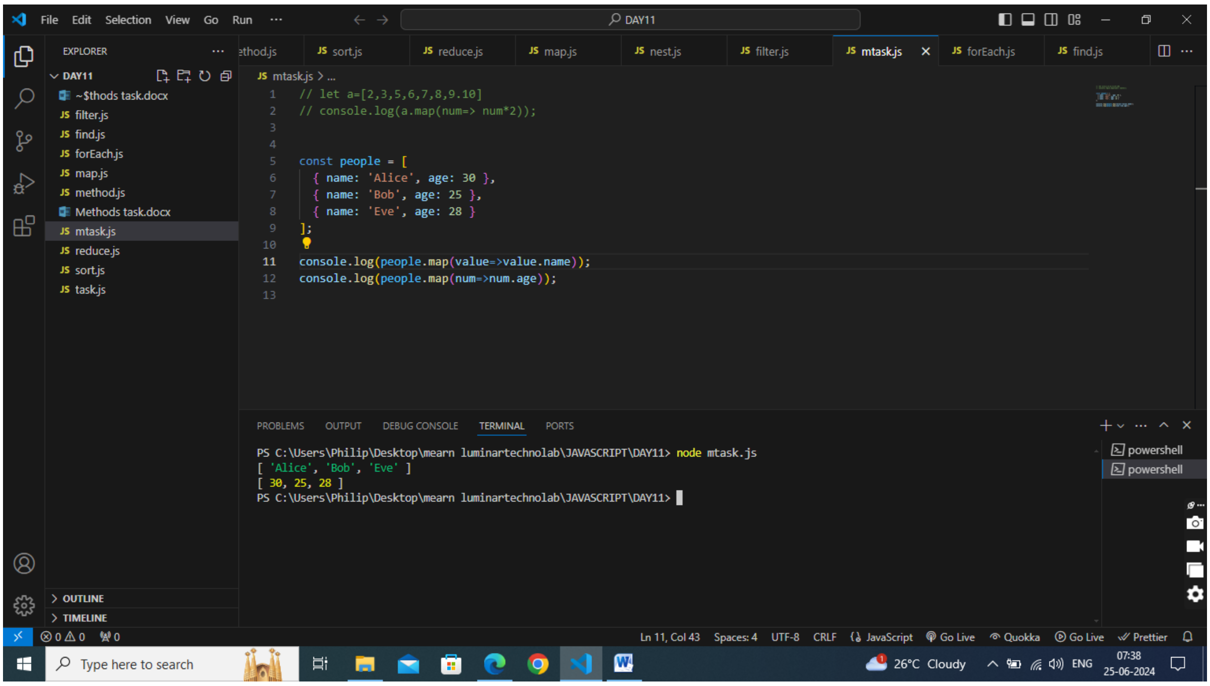
- ➢ map(): Used when needed to transform data and obtain a new array based on the transformation.
- ➢ forEach():Used to perform operations on each element without the need for a new array (e.g., logging, updating the original array, invoking side effects).

Original Array:

- ➢ map(): Does not modify the original array.
- ➢ forEach(): Does not inherently modify the original array, but the callback function can be used to do so.

//8 Can you demonstrate how to use the map() method to extract specific properties from an array of objects?

```
/*

const people = [

  { name: 'Alice', age: 30 },

  { name: 'Bob', age: 25 },

  { name: 'Eve', age: 28 }

];

*/
```



```
// let a=[2,3,5,6,7,8,9.10]
// console.log(a.map(num=> num*2));


const people = [
  { name: 'Alice', age: 30 },
  { name: 'Bob', age: 25 },
  { name: 'Eve', age: 28 }
];

console.log(people.map(value=>value.name));
console.log(people.map(num=>num.age));
```

```
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11> node mtask.js
[ 'Alice', 'Bob', 'Eve' ]
[ 30, 25, 28 ]
PS C:\Users\Philip\Desktop\mearn luminartechnolab\JAVASCRIPT\DAY11>
```

//9 How does the reduce() method work? Can you explain the basic idea behind its functionality?

It is used to execute a reducer function on each element of an array, resulting in a single output value. This method is particularly useful for accumulating values or processing arrays into a single value, such as a sum, product, or concatenated string. The reduce() method iterates through the array, applying the provided reducer function to each element along with an accumulator. The result of this function is stored in the accumulator and is passed to the next iteration.

Syntax : array.reduce(function(total, currentValue, currentIndex, arr), initialValue

Initialization: The reduce() method takes two main parameters:

A callback function which is applied to each element of the array.

An optional initial value (initialValue), which is used as the initial value for the accumulator (accumulator). If not provided, the first element of the array is used as the initial accumulator value and iteration starts from the second element.

Callback Function: The callback function accepts four parameters:

accumulator: The accumulator accumulates the callback's return values. It carries the accumulated result from previous iterations.

currentValue: The current element being processed in the array.

currentIndex (optional): The index of the current element being processed. array (optional): The array reduce was called upon.

Execution: The reduce() method executes the callback function once for each element present in the array (excluding undefined array elements), from left to right. It updates the accumulator with the return value of the callback function after each iteration.

Value: The reduce() method returns the accumulated result (accumulator) after the last iteration or the initial value if the array is empty and no initial value was provided.

//10 How does the reduceRight() method differ from the reduce() method?

reduce() Method

- The reduce() method processes the array from left to right, iterating over each element and accumulating a single result. Here's how it works:
- Processing Order: Starts from the first element (index 0) and moves towards the end of the array (array.length - 1).
- Callback Function: Accepts an accumulator that accumulates the callback's return values and a currentValue representing the current element being processed.
- Return Value: Returns the accumulated result after processing all elements of the array, optionally starting with an initial value.

reduceRight() Method

- The reduceRight() method processes the array from right to left, iterating over each element in reverse order:
- Processing Order: Starts from the last element (index array.length - 1) and moves towards the beginning of the array (0).
- Callback Function: Similar to reduce(), it accepts an accumulator and a currentValue, but processes elements from right to left.

- Return Value: Returns the accumulated result after processing all elements of the array in reverse order, optionally starting with an initial value