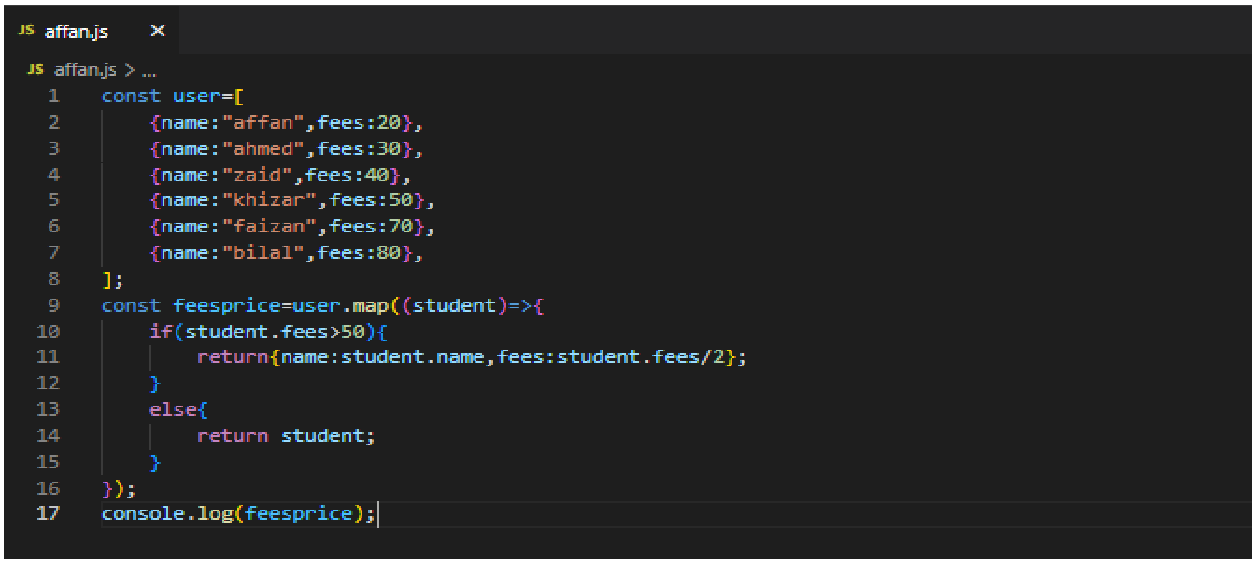
ASSIGNMENT:03
DAY:1,2

JAVA SCRIPT

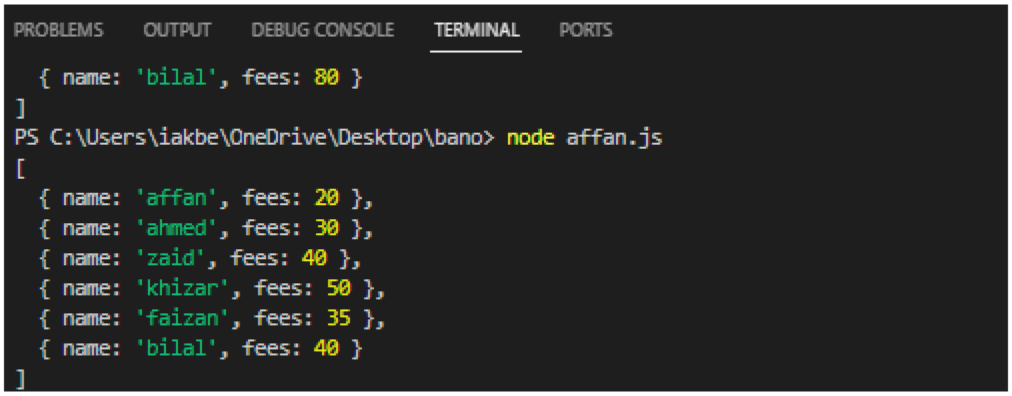
1)How does the map method work in JavaScript, and can you provide an example of when you might use it to manipulate an array of objects?

SOURCE CODE:



```javascript
const user=[
    {name:"affan",fees:20},
    {name:"ahmed",fees:30},
    {name:"zaid",fees:40},
    {name:"khizar",fees:50},
    {name:"faizan",fees:70},
    {name:"bilal",fees:80},
];
const feesprice=user.map((student)=>{
    if(student.fees>50){
        return{name:student.name,fees:student.fees/2};
    }
    else{
        return student;
    }
});
console.log(feesprice);
```

OUTPUT:



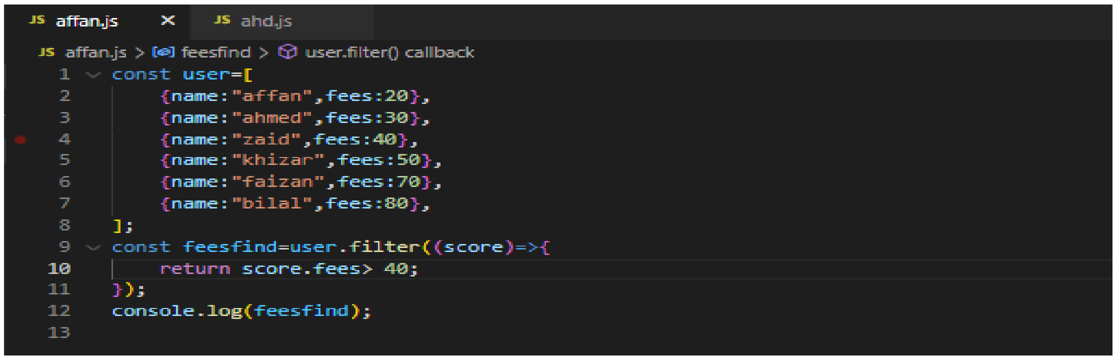
```
  { name: 'bilal', fees: 80 }
]
PS C:\Users\iakbe\OneDrive\Desktop\bano> node affan.js
[
  { name: 'affan', fees: 20 },
  { name: 'ahmed', fees: 30 },
  { name: 'zaid', fees: 40 },
  { name: 'khizar', fees: 50 },
  { name: 'faizan', fees: 35 },
  { name: 'bilal', fees: 40 }
]
```

ASSIGNMENT:03
DAY:1,2

2) *Filter Method:*

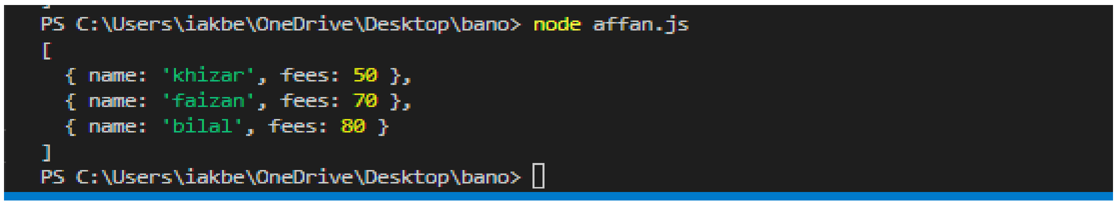
- Q: Explain the purpose of the filter method. Provide an example where you use filter to extract elements from an array based on a specific condition.

SOURCE CODE:



```
affan.js  ×   ahd.js
affan.js > feesfind > user.filter() callback
const user=[
    {name:"affan",fees:20},
    {name:"ahmed",fees:30},
    {name:"zaid",fees:40},
    {name:"khizar",fees:50},
    {name:"faizan",fees:70},
    {name:"bilal",fees:80},
];
const feesfind=user.filter((score)=>{
    return score.fees> 40;
});
console.log(feesfind);
```

OUTPUT:

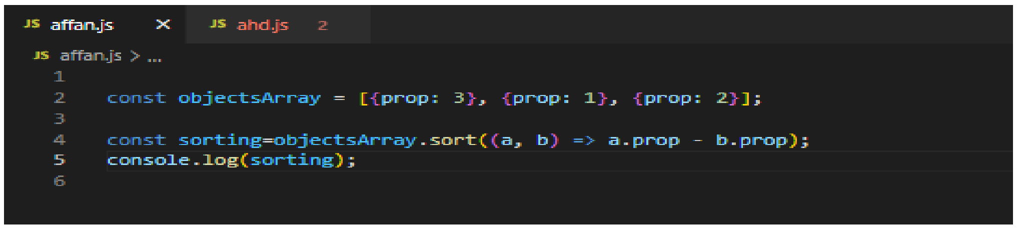


```
PS C:\Users\iakbe\OneDrive\Desktop\bano> node affan.js
[
  { name: 'khizar', fees: 50 },
  { name: 'faizan', fees: 70 },
  { name: 'bilal', fees: 80 }
]
PS C:\Users\iakbe\OneDrive\Desktop\bano>
```

3) *Sort Method:*

- Q: Discuss the default behavior of the sort method for strings and numbers. How would you use a custom comparison function to sort an array of objects by a specific property?

SOURCE CODE:



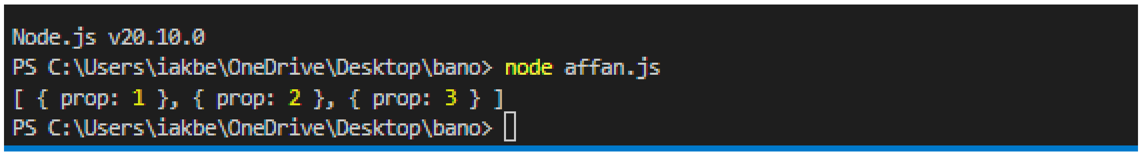
```
affan.js  ×   ahd.js  2
affan.js > ...

const objectsArray = [{prop: 3}, {prop: 1}, {prop: 2}];

const sorting=objectsArray.sort((a, b) => a.prop - b.prop);
console.log(sorting);

```

OUTPUT:



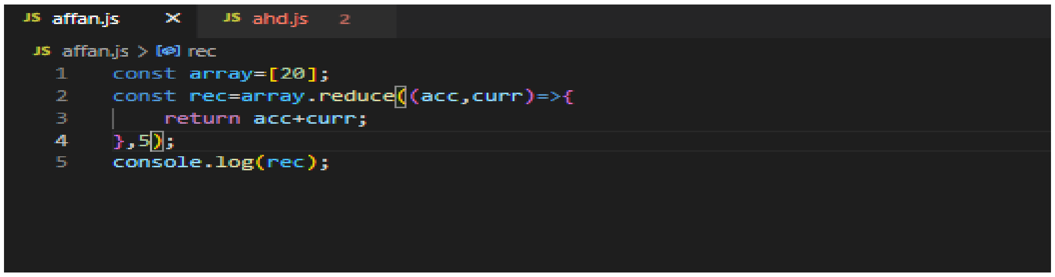
```
Node.js v20.10.0
PS C:\Users\iakbe\OneDrive\Desktop\bano> node affan.js
[ { prop: 1 }, { prop: 2 }, { prop: 3 } ]
PS C:\Users\iakbe\OneDrive\Desktop\bano>
```

ASSIGNMENT:03
DAY:1,2

4.)*Reduce Method:*

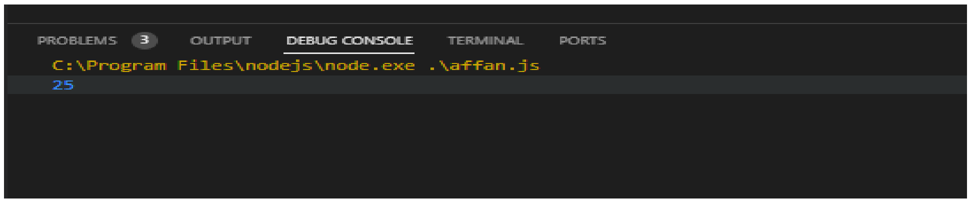
- Q: Describe the purpose of the reduce method and provide an example where you use it to compute a single value from an array of numbers.

SOURCE CODE:



```
const array=[20];
const rec=array.reduce((acc,curr)=>{
    return acc+curr;
},5);
console.log(rec);
```

OUTPUT:

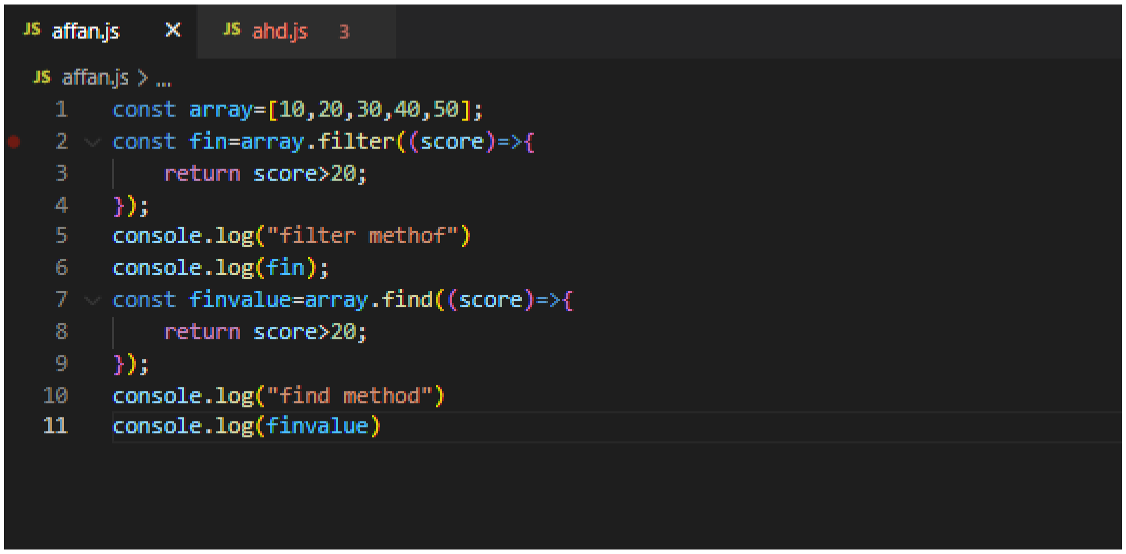


```
C:\Program Files\nodejs\node.exe .\affan.js
25
```

5.)*Find Method:*

- Q: How does the find method differ from filter? Give an example of a scenario where using find is more appropriate than filter.

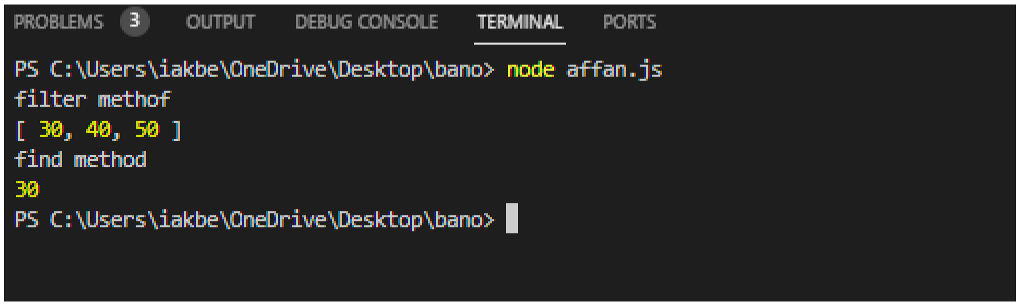
SOURCE CODE:



```
const array=[10,20,30,40,50];
const fin=array.filter((score)=>{
    return score>20;
});
console.log("filter methof")
console.log(fin);
const finvalue=array.find((score)=>{
    return score>20;
});
console.log("find method")
console.log(finvalue)
```

ASSIGNMENT:03
DAY:1,2

OUTPUT:

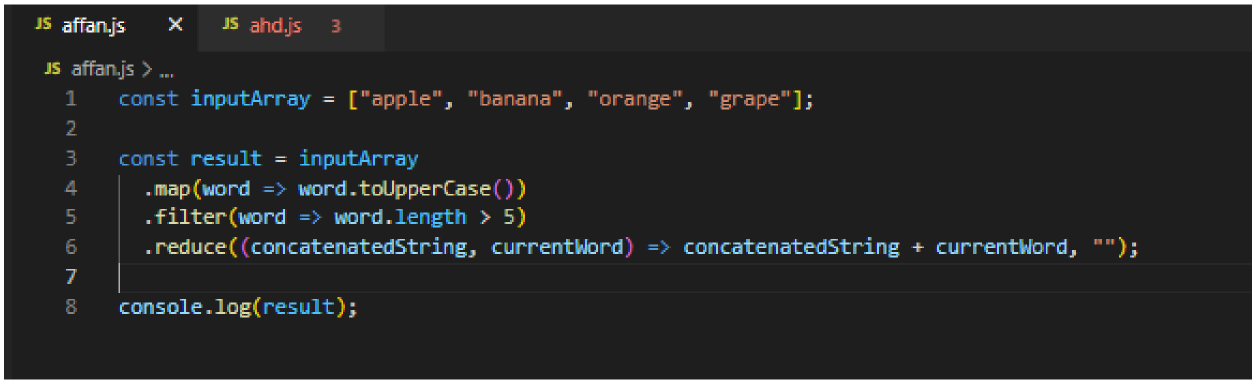


```
PS C:\Users\iakbe\OneDrive\Desktop\bano> node affan.js
filter methof
[ 30, 40, 50 ]
find method
30
PS C:\Users\iakbe\OneDrive\Desktop\bano>
```

6) *Combining Methods:*

- Q: Create a chain of array methods (map, filter, reduce, etc.) to transform an array of strings into a single concatenated string with a specific condition.

SOURCE CODE:

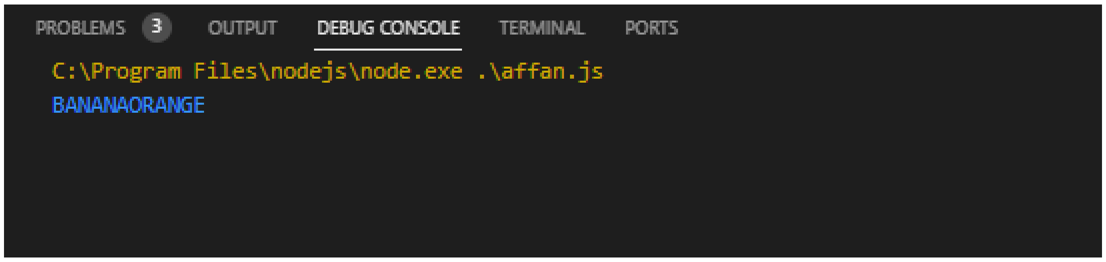


```javascript
const inputArray = ["apple", "banana", "orange", "grape"];

const result = inputArray
  .map(word => word.toUpperCase())
  .filter(word => word.length > 5)
  .reduce((concatenatedString, currentWord) => concatenatedString + currentWord, "");

console.log(result);
```

OUTPUT:



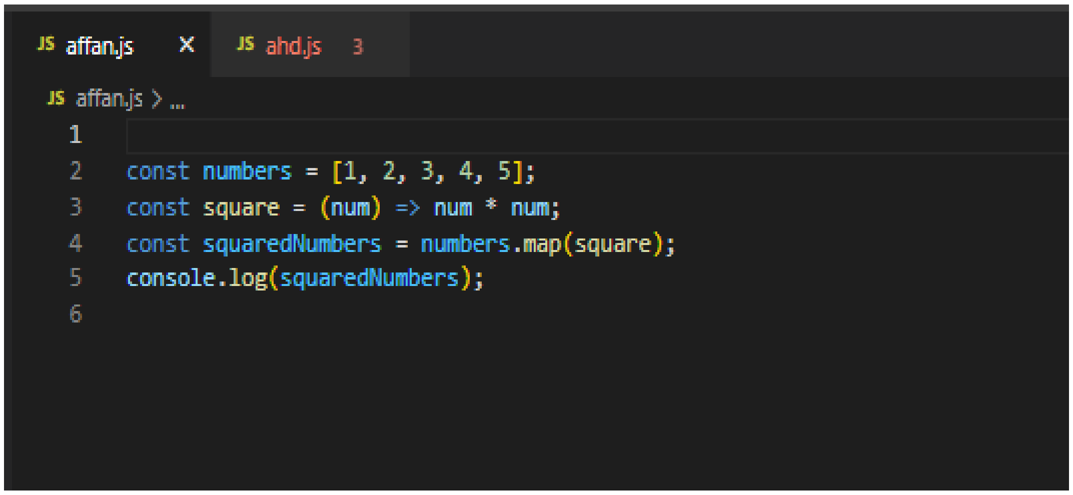
```
C:\Program Files\nodejs\node.exe .\affan.js
BANANAORANGE
```

ASSIGNMENT:03
DAY:1,2

7. *Callback Functions:*

- Q: Explain the concept of callback functions in the context of array methods. Provide an example of using a callback function with the map method.

SOURCE CODE:

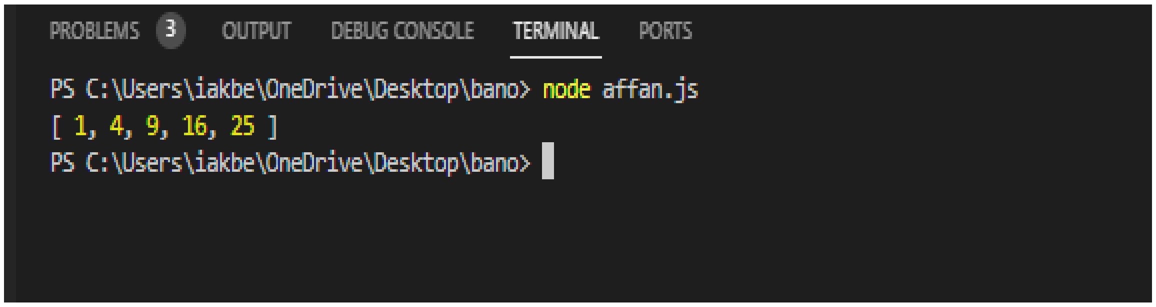


```

const numbers = [1, 2, 3, 4, 5];
const square = (num) => num * num;
const squaredNumbers = numbers.map(square);
console.log(squaredNumbers);

```

OUTPUT:



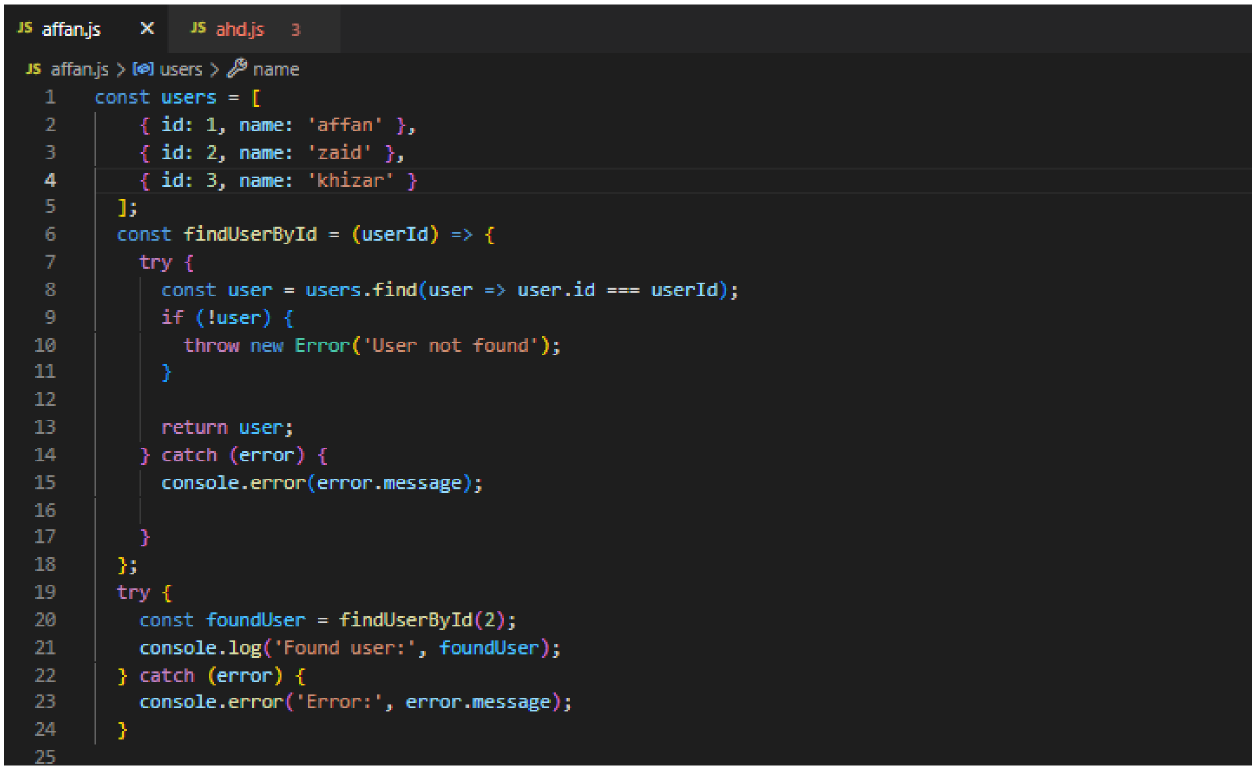
```
PS C:\Users\iakbe\OneDrive\Desktop\bano> node affan.js
[ 1, 4, 9, 16, 25 ]
PS C:\Users\iakbe\OneDrive\Desktop\bano>
```

ASSIGNMENT:03
DAY:1,2

8) *Error Handling:*

- Q: How would you handle potential errors when using array methods like find or reduce? Provide an example of error handling in such a scenario.

SOURCE CODE:

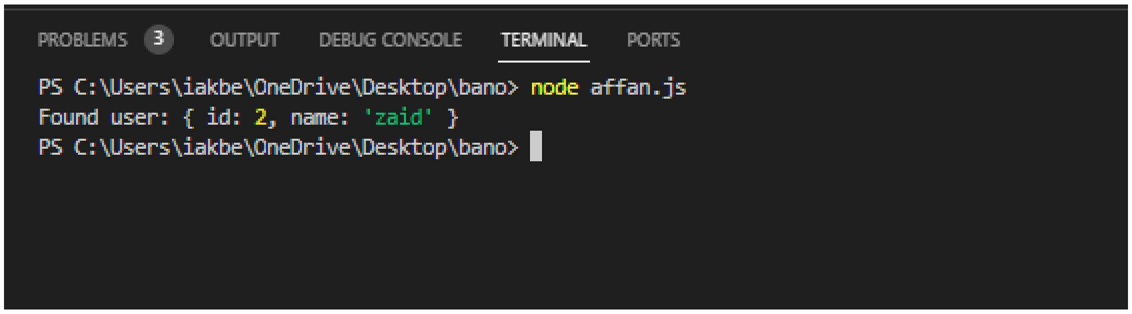

affan.js ✕ ahd.js 3
affan.js > users > name

```javascript
const users = [
    { id: 1, name: 'affan' },
    { id: 2, name: 'zaid' },
    { id: 3, name: 'khizar' }
  ];
  const findUserById = (userId) => {
    try {
      const user = users.find(user => user.id === userId);
      if (!user) {
        throw new Error('User not found');
      }

      return user;
    } catch (error) {
      console.error(error.message);

    }
  };
  try {
    const foundUser = findUserById(2);
    console.log('Found user:', foundUser);
  } catch (error) {
    console.error('Error:', error.message);
  }
```

OUTPUT:



```
PS C:\Users\iakbe\OneDrive\Desktop\bano> node affan.js
Found user: { id: 2, name: 'zaid' }
PS C:\Users\iakbe\OneDrive\Desktop\bano>
```

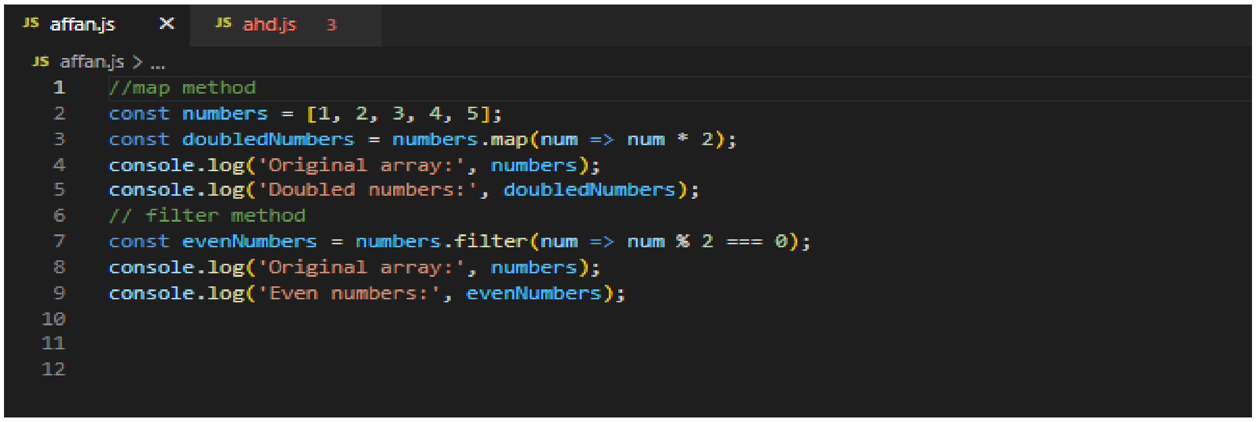
ASSIGNMENT:03

DAY:1,2

9. *Immutable Operations:*

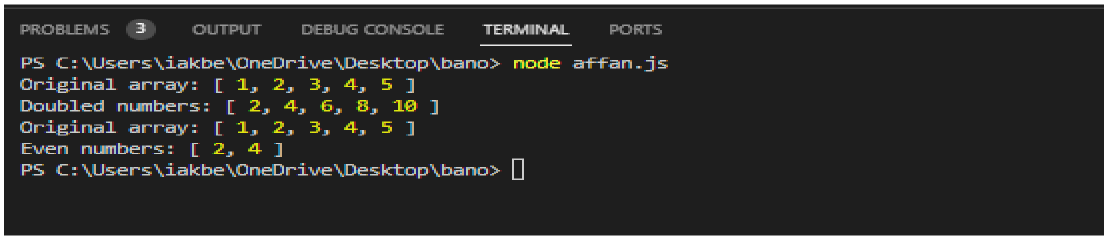
- Q: Discuss the importance of immutability when working with array methods. Demonstrate how you would perform immutable operations using methods like map or filter.

SOURCE CODE:



```javascript
//map method
const numbers = [1, 2, 3, 4, 5];
const doubledNumbers = numbers.map(num => num * 2);
console.log('Original array:', numbers);
console.log('Doubled numbers:', doubledNumbers);
// filter method
const evenNumbers = numbers.filter(num => num % 2 === 0);
console.log('Original array:', numbers);
console.log('Even numbers:', evenNumbers);
```

OUTPUT:

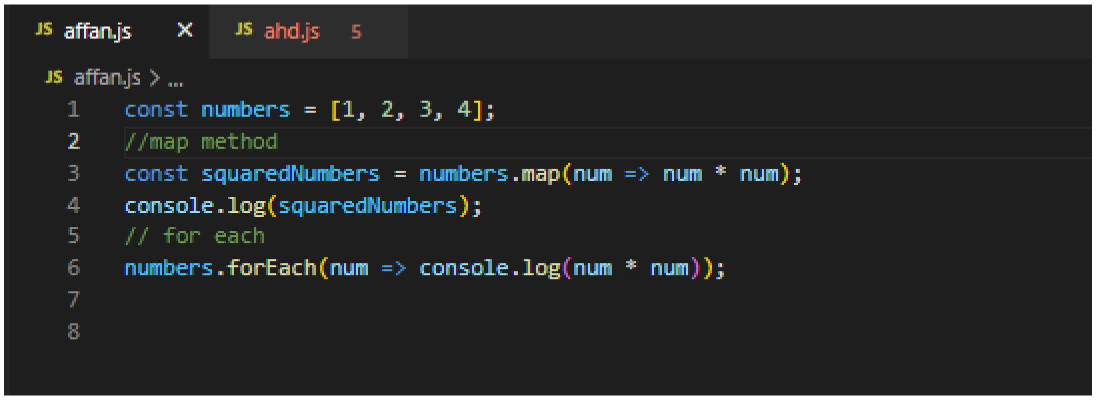


```
PS C:\Users\iakbe\OneDrive\Desktop\bano> node affan.js
Original array: [ 1, 2, 3, 4, 5 ]
Doubled numbers: [ 2, 4, 6, 8, 10 ]
Original array: [ 1, 2, 3, 4, 5 ]
Even numbers: [ 2, 4 ]
PS C:\Users\iakbe\OneDrive\Desktop\bano>
```

10. *Performance Considerations:*

- Q: Compare the performance implications of using map versus forEach. In what scenarios would you prefer one over the other, and why?

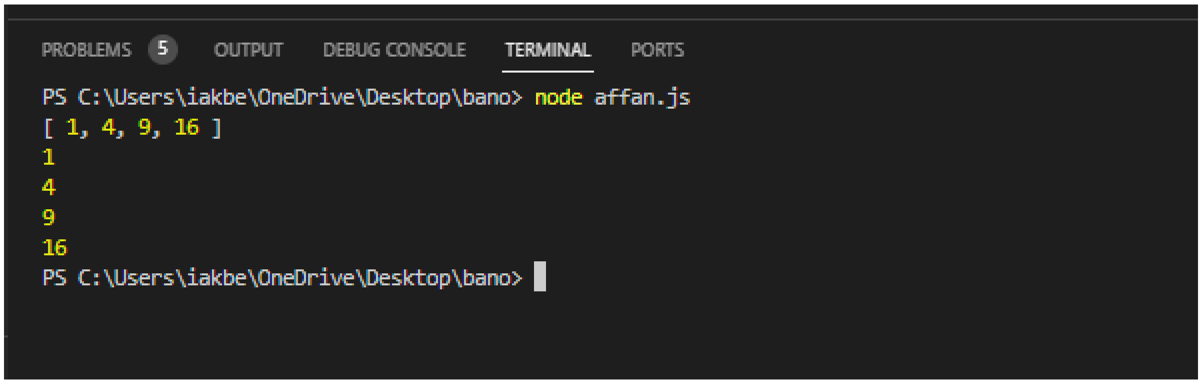
SOURCE CODE:



```javascript
const numbers = [1, 2, 3, 4];
//map method
const squaredNumbers = numbers.map(num => num * num);
console.log(squaredNumbers);
// for each
numbers.forEach(num => console.log(num * num));
```

ASSIGNMENT:03
DAY:1,2

OUTPUT:



PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\iakbe\OneDrive\Desktop\bano> node affan.js
[ 1, 4, 9, 16 ]
1
4
9
16
PS C:\Users\iakbe\OneDrive\Desktop\bano>
```