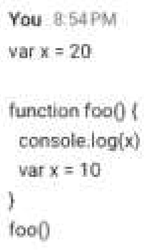
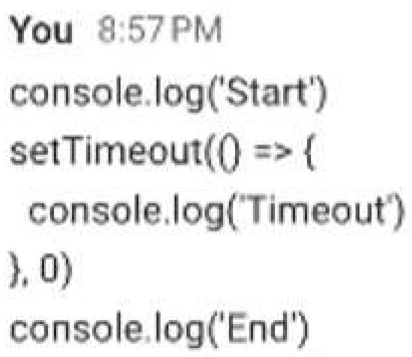
snippets



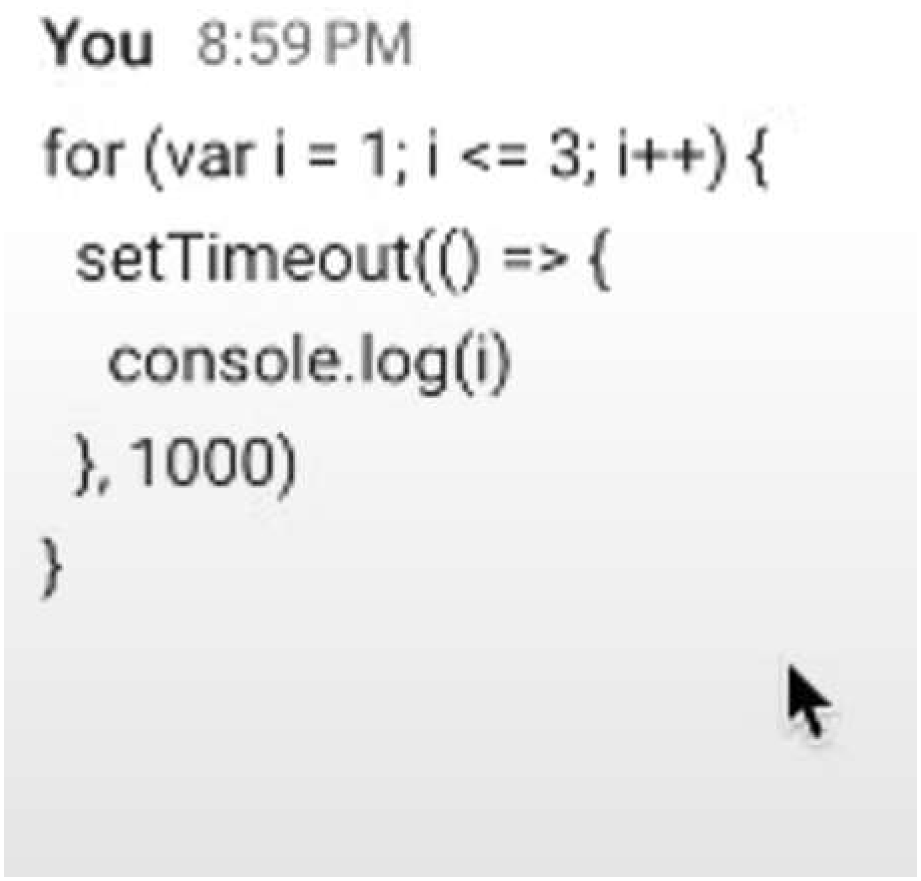
You 8:54 PM

```
var x = 20

function foo() {
  console.log(x)
  var x = 10
}
foo()
```



You 8:57 PM

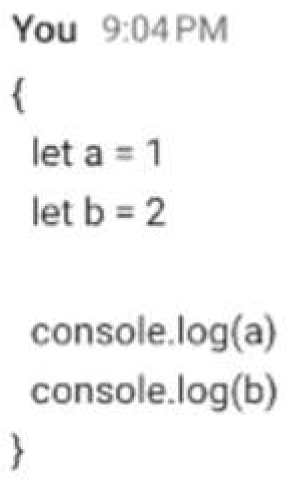
```
console.log('Start')
setTimeout(() => {
  console.log('Timeout')
}, 0)
console.log('End')
```



You 8:59 PM

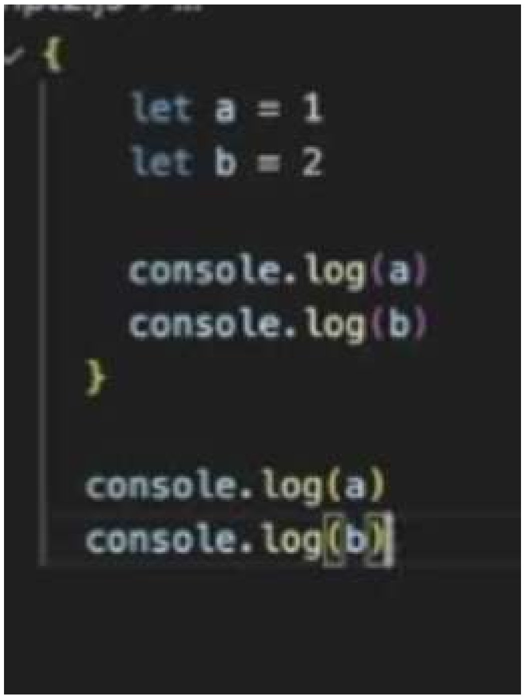
```
for (var i = 1; i <= 3; i++) {
 setTimeout(() => {
  console.log(i)
 }, 1000)
}
```

3 times 4

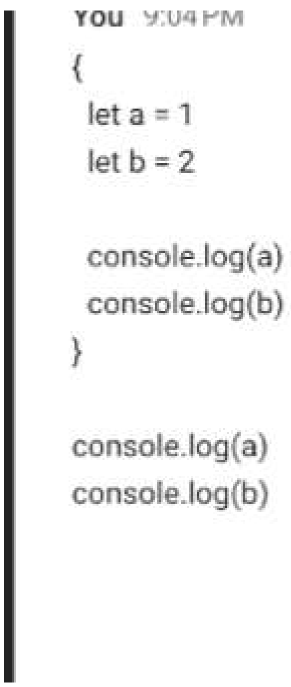


You 9:04 PM

```
{
 let a = 1
 let b = 2

 console.log(a)
 console.log(b)
}
```

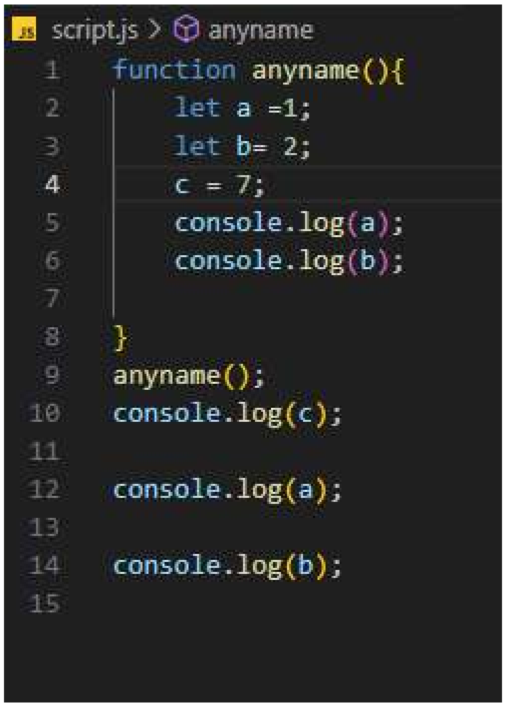


```
{
    let a = 1
    let b = 2

    console.log(a)
    console.log(b)
 }

console.log(a)
console.log(b)
```

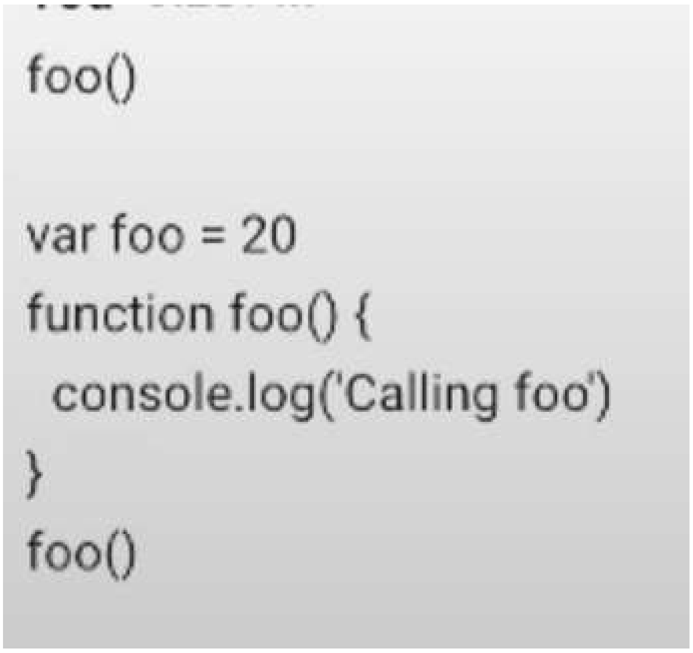


You 9:04 PM

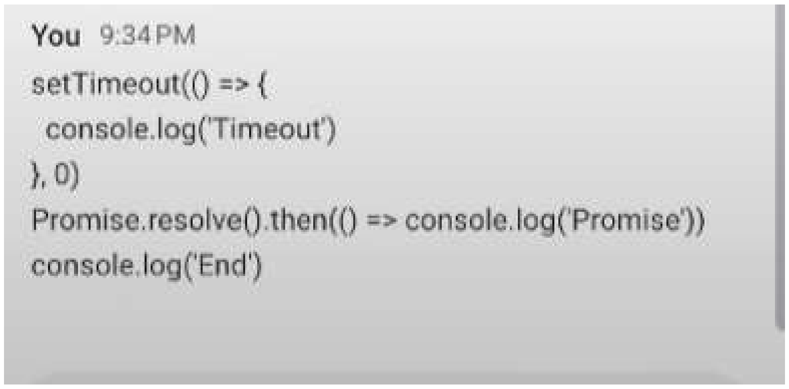
```
{
  let a = 1
  let b = 2

  console.log(a)
  console.log(b)
}

console.log(a)
console.log(b)
```



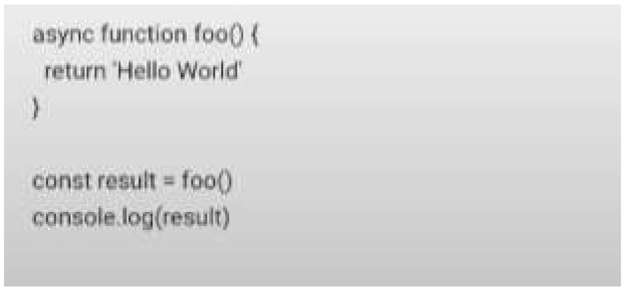
```js
function anyname(){
    let a =1;
    let b= 2;
    c = 7;
    console.log(a);
    console.log(b);

}
anyname();
console.log(c);

console.log(a);

console.log(b);

```



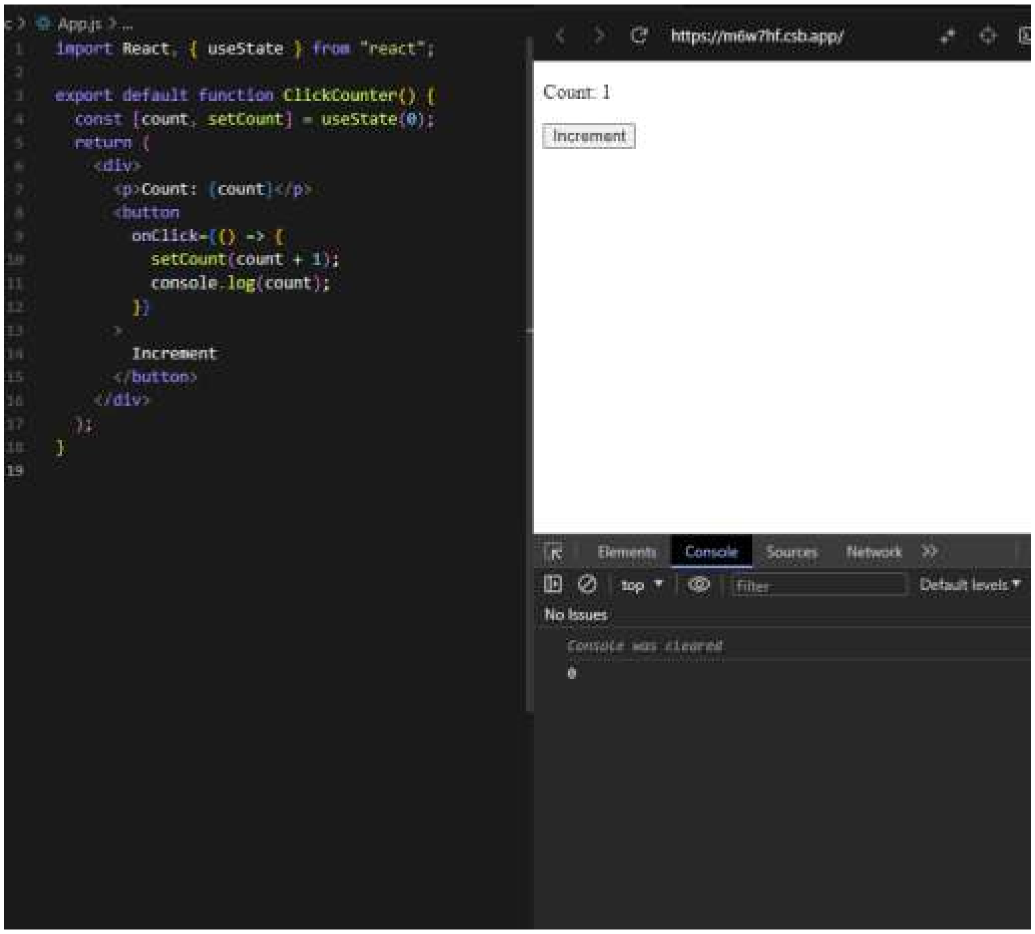
```js
foo()


var foo = 20
function foo() {
  console.log('Calling foo')
}
foo()
```


You 9:34 PM

```
setTimeout(() => {
  console.log('Timeout')
}, 0)
Promise.resolve().then(() => console.log('Promise'))
console.log('End')
```



```
async function foo() {
  return 'Hello World'
}

const result = foo()
console.log(result)
```



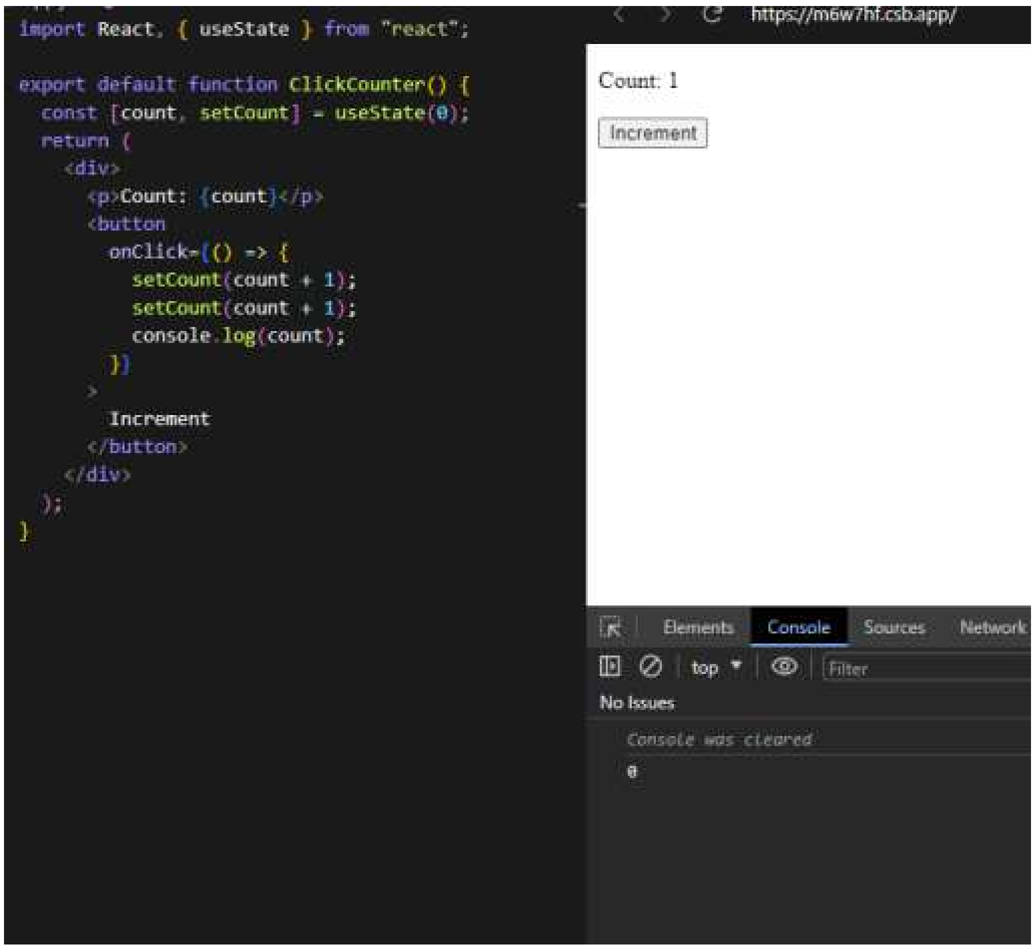
```
import React, { useState } from "react";

export default function ClickCounter() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>Count: {count}</p>
      <button
        onClick={() => {
          setCount(count + 1);
          console.log(count);
        }}
      >
        Increment
      </button>
    </div>
  );
}
```

https://m6w7hf.csb.app/

Count: 1

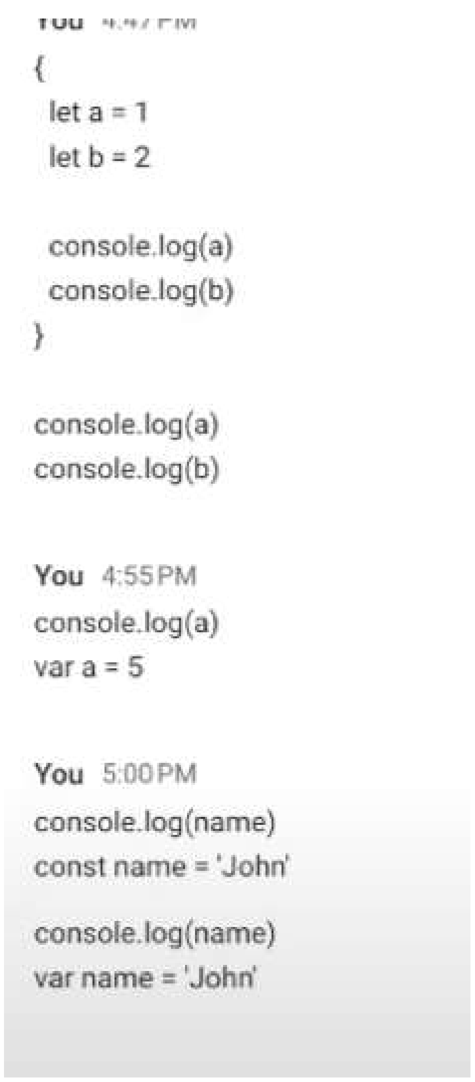
Increment

```
Console was cleared
0
```



```
import React, { useState } from "react";

export default function ClickCounter() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>Count: {count}</p>
      <button
        onClick={() => {
          setCount(count + 1);
          setCount(count + 1);
          console.log(count);
        }}
      >
        Increment
      </button>
    </div>
  );
}
```

```
Console was cleared
0
```



You 4:47 PM

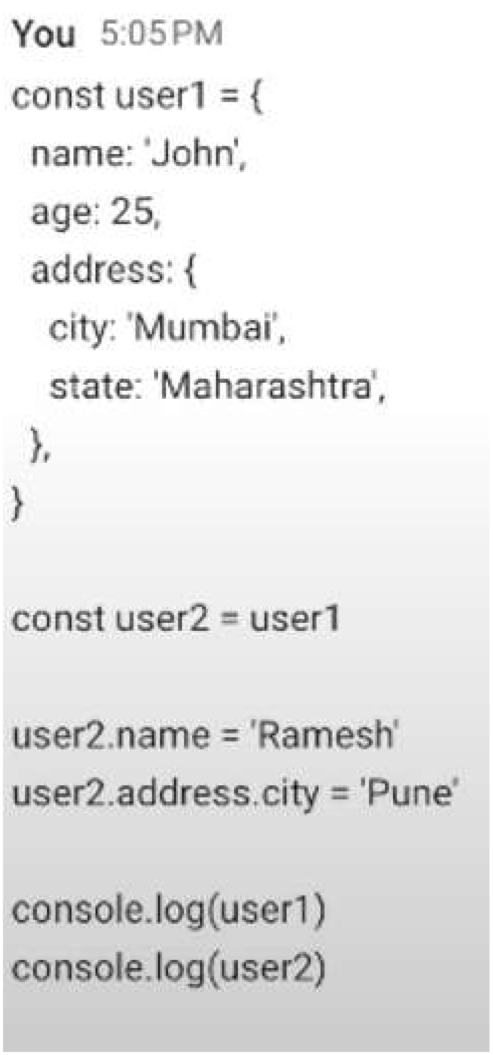
```
{
 let a = 1
 let b = 2

 console.log(a)
 console.log(b)
}

console.log(a)
console.log(b)
```

You 4:55 PM

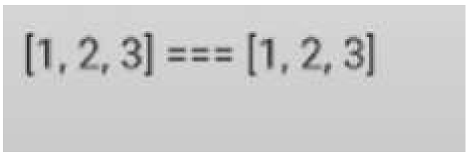
```
console.log(a)
var a = 5
```

You 5:00 PM

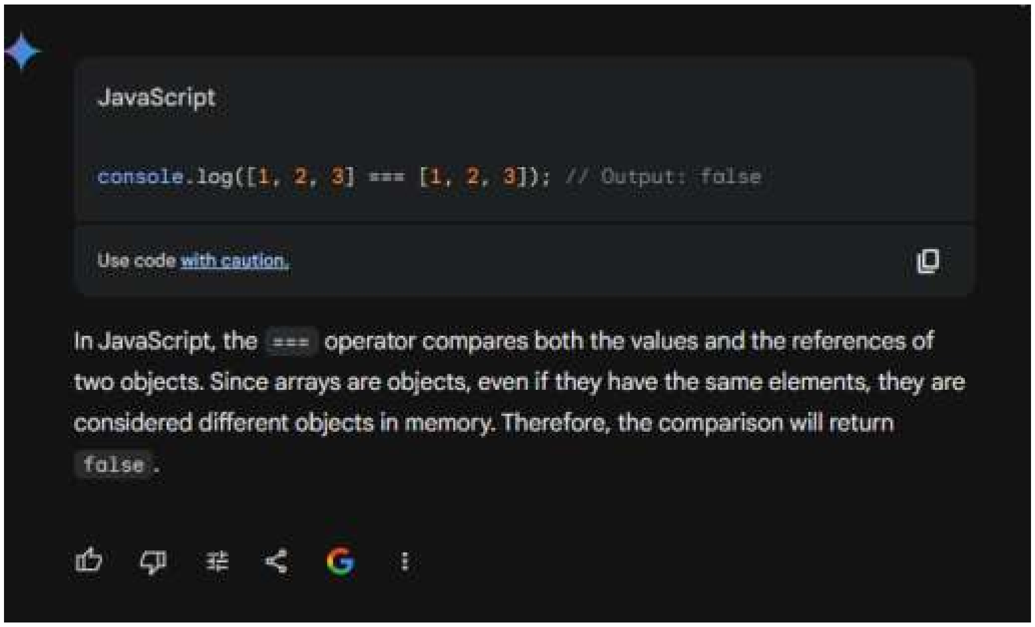
```
console.log(name)
const name = 'John'

console.log(name)
var name = 'John'
```


You 5:05PM

```
const user1 = {
  name: 'John',
  age: 25,
  address: {
    city: 'Mumbai',
    state: 'Maharashtra',
  },
}

const user2 = user1

user2.name = 'Ramesh'
user2.address.city = 'Pune'

console.log(user1)
console.log(user2)
```

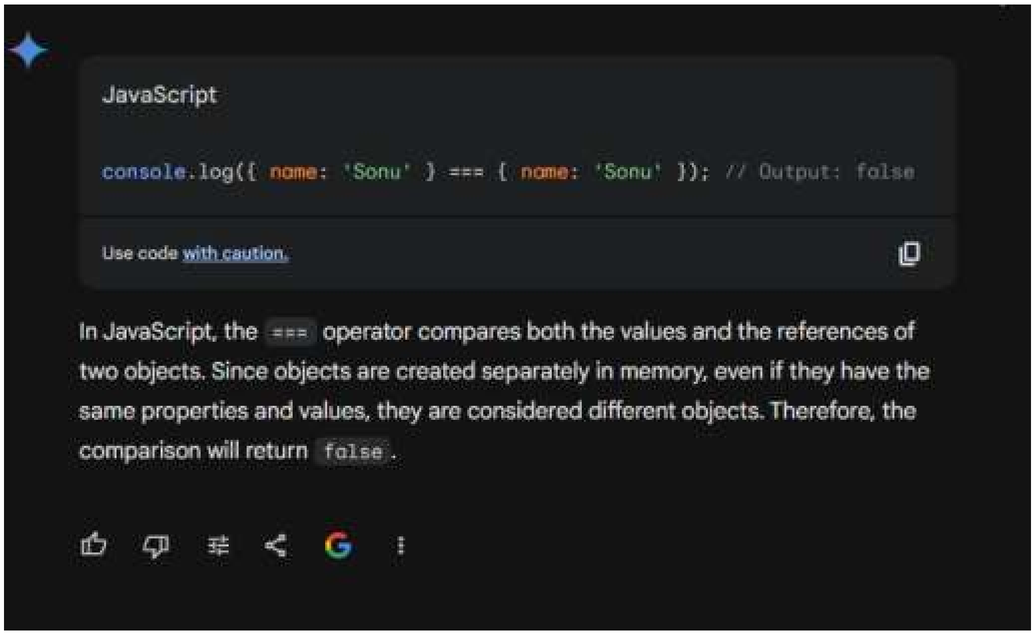


```
[1, 2, 3] === [1, 2, 3]
```



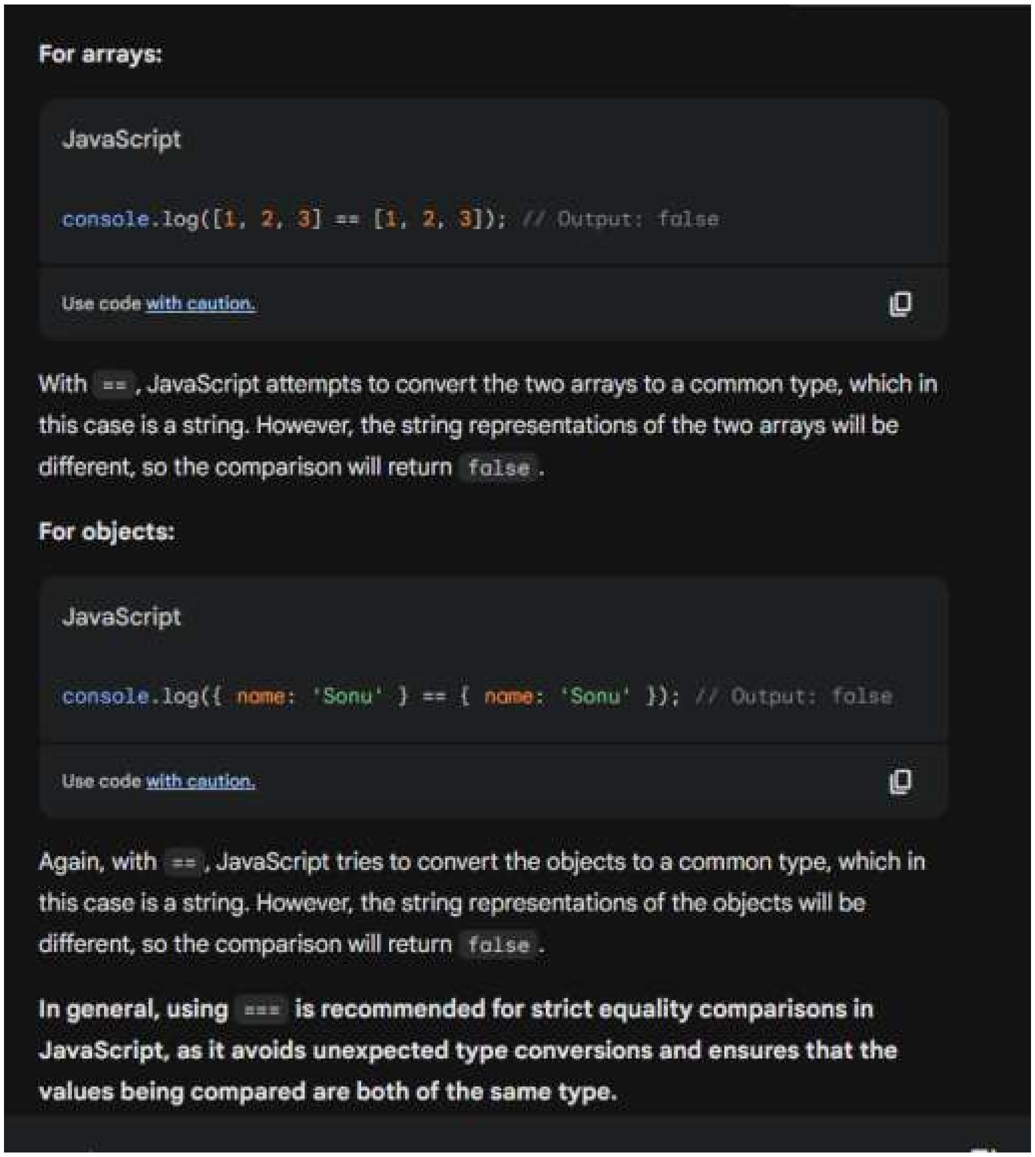
```
console.log([1, 2, 3] === [1, 2, 3]); // Output: false
```

In JavaScript, the `===` operator compares both the values and the references of two objects. Since arrays are objects, even if they have the same elements, they are considered different objects in memory. Therefore, the comparison will return `false`.



```
console.log({ name: 'Sonu' } === { name: 'Sonu' }); // Output: false
```

In JavaScript, the `===` operator compares both the values and the references of two objects. Since objects are created separately in memory, even if they have the same properties and values, they are considered different objects. Therefore, the comparison will return `false`.



For arrays:

JavaScript

```
console.log([1, 2, 3] == [1, 2, 3]); // Output: false
```

Use code with caution.

With `==`, JavaScript attempts to convert the two arrays to a common type, which in this case is a string. However, the string representations of the two arrays will be different, so the comparison will return `false`.

For objects:

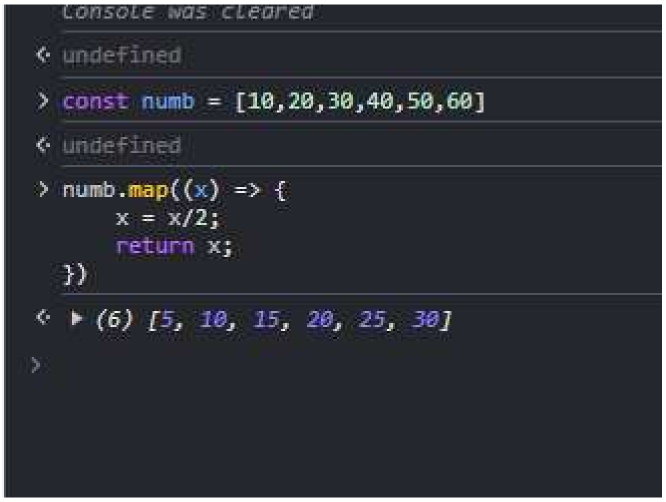
JavaScript

```
console.log({ name: 'Sonu' } == { name: 'Sonu' }); // Output: false
```

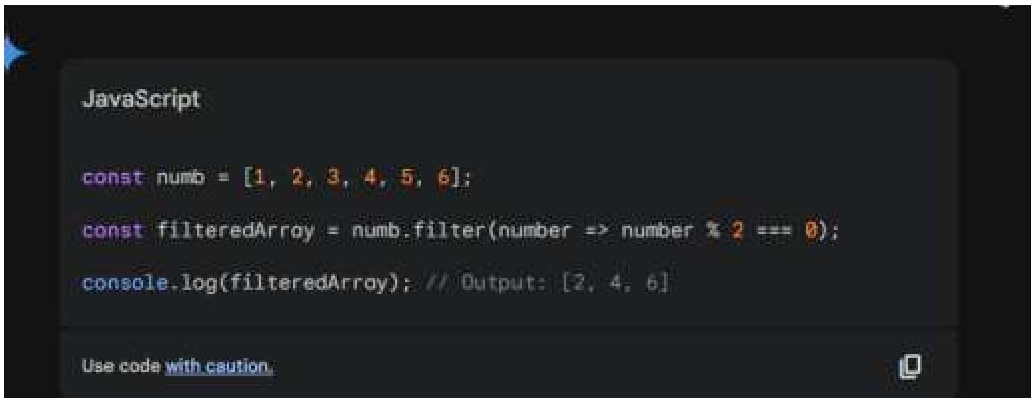
Use code with caution.

Again, with `==`, JavaScript tries to convert the objects to a common type, which in this case is a string. However, the string representations of the objects will be different, so the comparison will return `false`.

In general, using `===` is recommended for strict equality comparisons in JavaScript, as it avoids unexpected type conversions and ensures that the values being compared are both of the same type.



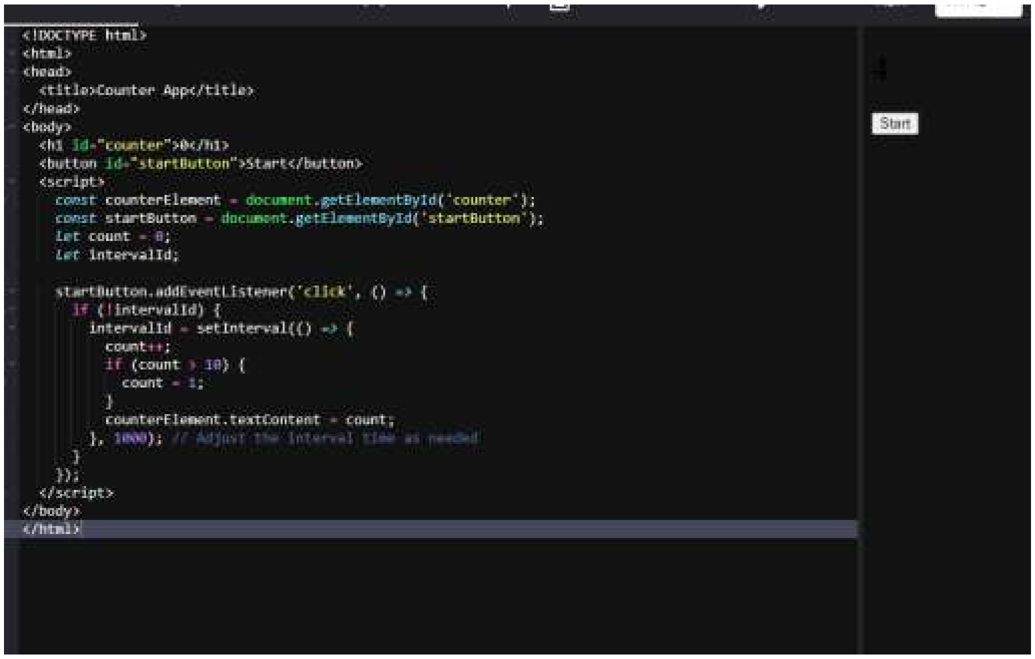
```
  Console was cleared
< undefined
> const numb = [10,20,30,40,50,60]
< undefined
> numb.map((x) => {
      x = x/2;
      return x;
  })
< ▶ (6) [5, 10, 15, 20, 25, 30]
>
```



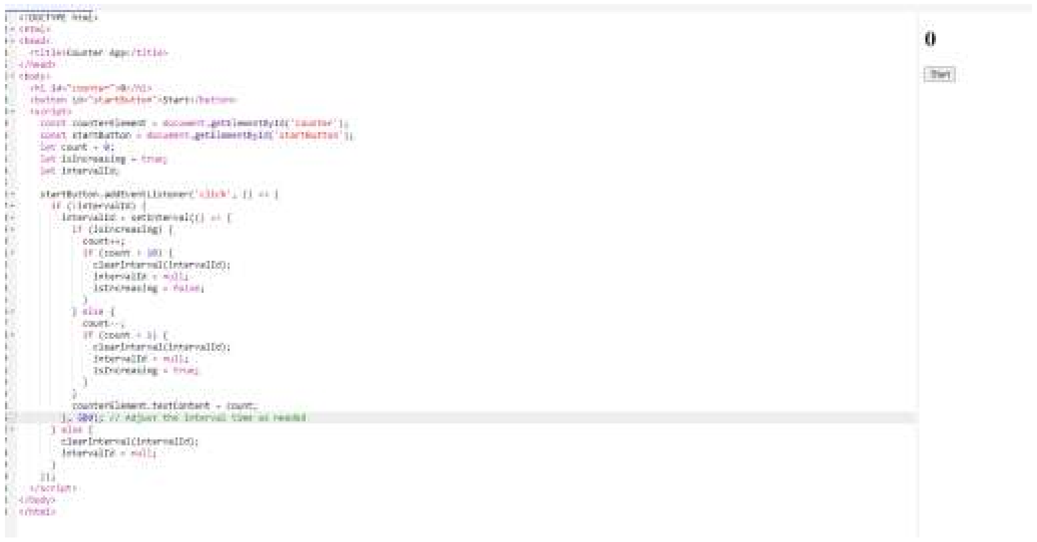
JavaScript

```javascript
const numb = [1, 2, 3, 4, 5, 6];

const filteredArray = numb.filter(number => number % 2 === 0);

console.log(filteredArray); // Output: [2, 4, 6]
```

Use code with caution.

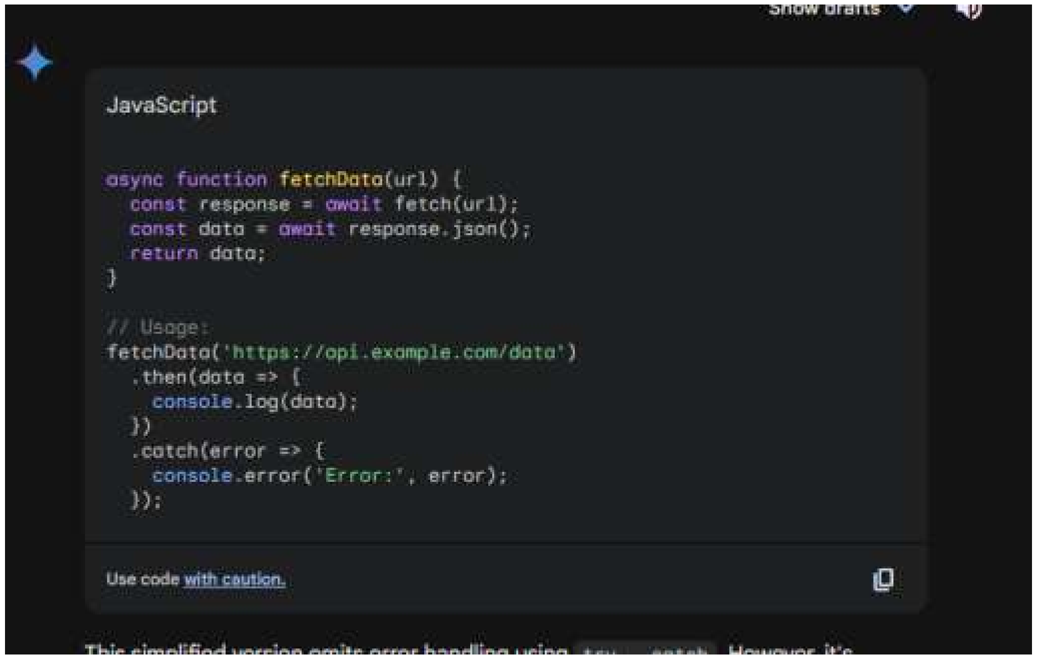


```html
<!DOCTYPE html>
<html>
<head>
  <title>Counter App</title>
</head>
<body>
  <h1 id="counter">0</h1>
  <button id="startButton">Start</button>
  <script>
    const counterElement = document.getElementById('counter');
    const startButton = document.getElementById('startButton');
    let count = 0;
    let intervalId;

    startButton.addEventListener('click', () => {
      if (!intervalId) {
        intervalId = setInterval(() => {
          count++;
          if (count > 10) {
            count = 1;
          }
          counterElement.textContent = count;
        }, 1000); // Adjust the interval time as needed
      }
    });
  </script>
</body>
</html>
```



```
<!DOCTYPE html>
<html>
<head>
  <title>Counter App</title>
</head>
<body>
  <h1 id="counter">0</h1>
  <button id="startButton">Start</button>
  <script>
    const counterElement = document.getElementById('counter');
    const startButton = document.getElementById('startButton');
    let count = 0;
    let isIncreasing = true;
    let intervalId;

    startButton.addEventListener('click', () => {
      if (!intervalId) {
        intervalId = setInterval(() => {
          if (isIncreasing) {
            count++;
            if (count > 10) {
              clearInterval(intervalId);
              intervalId = null;
              isIncreasing = false;
            }
          } else {
            count--;
            if (count < 1) {
              clearInterval(intervalId);
              intervalId = null;
              isIncreasing = true;
            }
          }
          counterElement.textContent = count;
        }, 500); // Adjust the interval time as needed
      } else {
        clearInterval(intervalId);
        intervalId = null;
      }
    });
  </script>
</body>
</html>
```

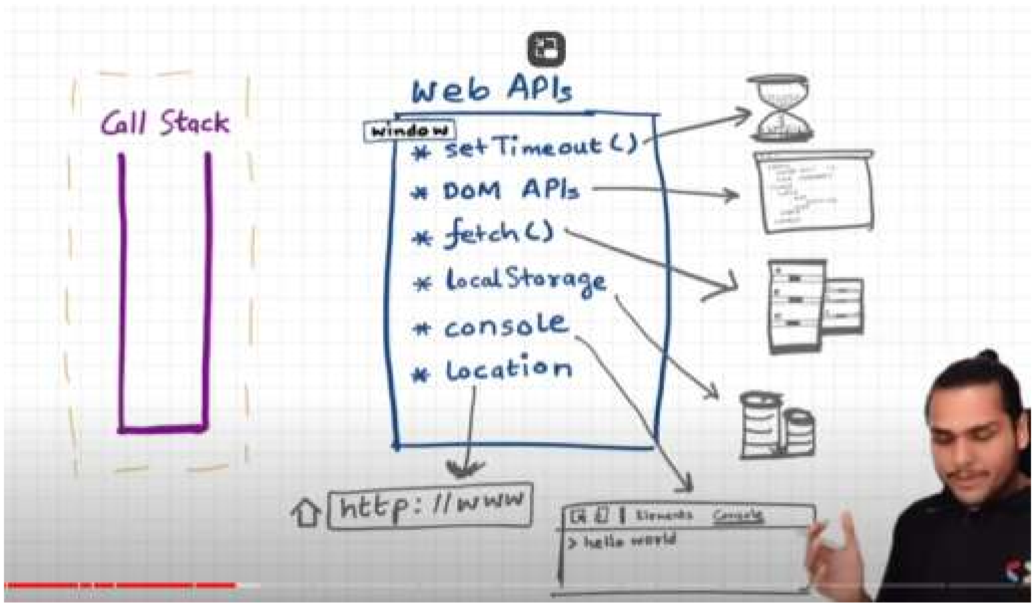
```
 function capitalizeFirstLetter(string) {
  return string.split(' ').map(word => word.charAt(0).toUpperCase() + word.slice(1)).join(' ');
}

// Example usage:
const text = "hello world this is a test";
const capitalizedText = capitalizeFirstLetter(text);
console.log(capitalizedText); // Output: Hello World This Is A Test
```



```
async function fetchData(url) {
  const response = await fetch(url);
  const data = await response.json();
  return data;
}

// Usage:
fetchData('https://api.example.com/data')
  .then(data => {
    console.log(data);
  })
  .catch(error => {
    console.error('Error:', error);
  });
```

Use code with caution.

This simplified version omits error handling using try...catch. However, it's


Call Stack
Web APIs
window
* setTimeout()
* DOM APIs
* fetch()
* localStorage
* console
* location
http://www
Elements Console
> hello world