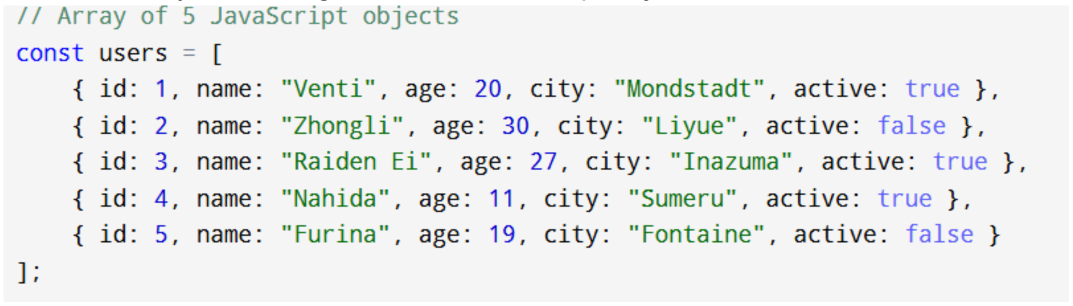
WORKING WITH JAVASCRIPT OBJECTS AND ASYNCHRONOUS PROGRAMMING

NAME: Navelgas, Quenzzy J.
SECTION: UCOS 4-2

INSTRUCTIONS: Please compile all the JavaScript codes in a Github repository and include it in the document. Paste **ANSWER AS SCREENSHOT** from programiz.com or any text editor **BELOW EACH ITEM**.

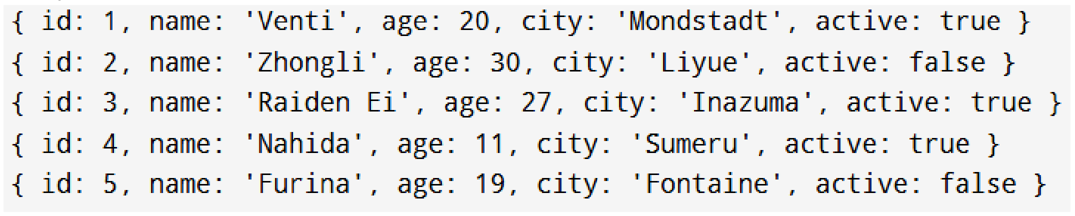
I. JAVASCRIPT OBJECTS:

A. Make an array containing at least 5 JavaScript objects.



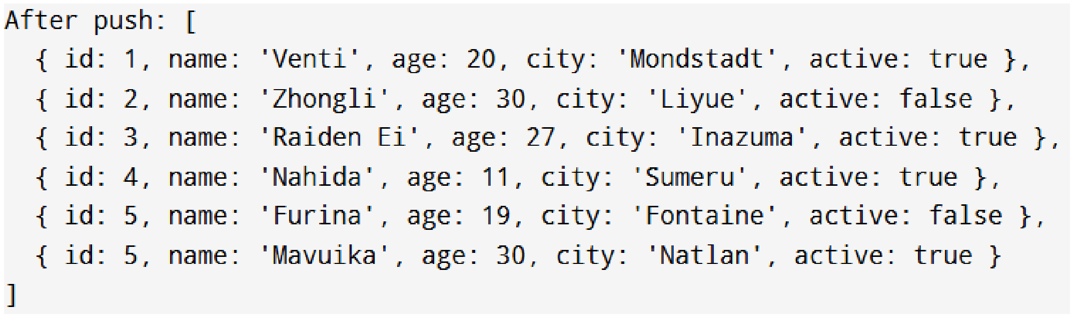
```
// Array of 5 JavaScript objects
const users = [
    { id: 1, name: "Venti", age: 20, city: "Mondstadt", active: true },
    { id: 2, name: "Zhongli", age: 30, city: "Liyue", active: false },
    { id: 3, name: "Raiden Ei", age: 27, city: "Inazuma", active: true },
    { id: 4, name: "Nahida", age: 11, city: "Sumeru", active: true },
    { id: 5, name: "Furina", age: 19, city: "Fontaine", active: false }
];
```

B. Use the same array and use the forEach() method to print each object in the array.



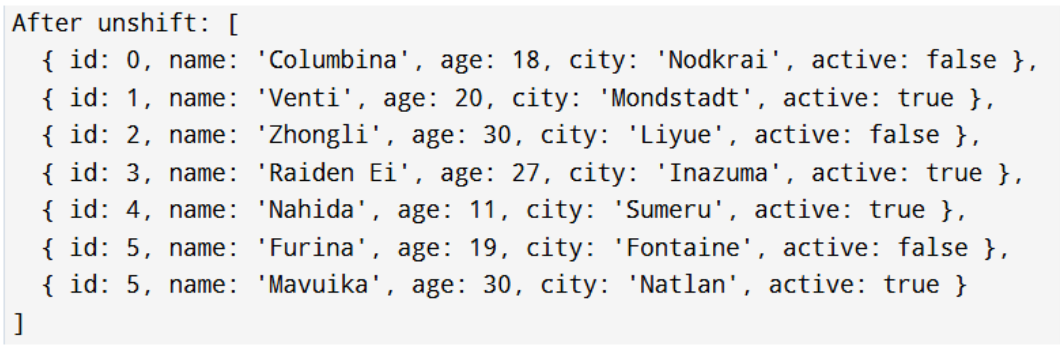
```
{ id: 1, name: 'Venti', age: 20, city: 'Mondstadt', active: true }
{ id: 2, name: 'Zhongli', age: 30, city: 'Liyue', active: false }
{ id: 3, name: 'Raiden Ei', age: 27, city: 'Inazuma', active: true }
{ id: 4, name: 'Nahida', age: 11, city: 'Sumeru', active: true }
{ id: 5, name: 'Furina', age: 19, city: 'Fontaine', active: false }
```

C. Use the same array and demonstrate a sample code using the push() method on the array.



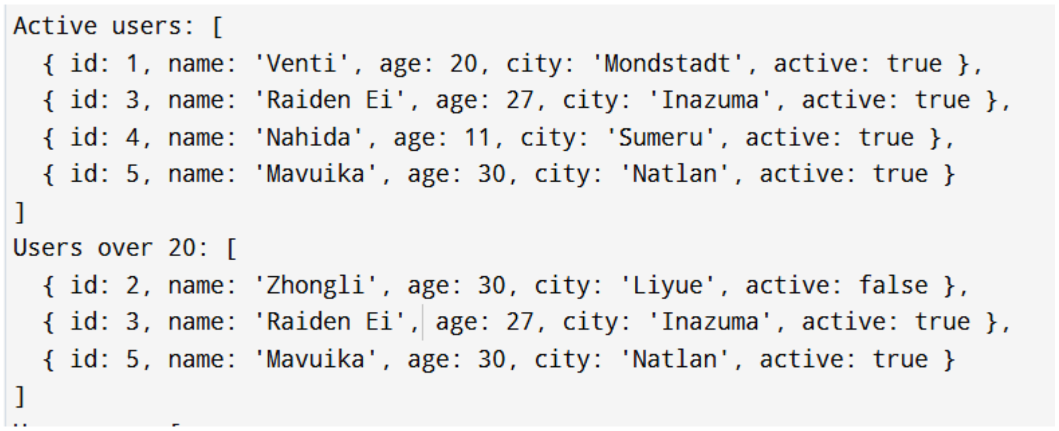
```
After push: [
  { id: 1, name: 'Venti', age: 20, city: 'Mondstadt', active: true },
  { id: 2, name: 'Zhongli', age: 30, city: 'Liyue', active: false },
  { id: 3, name: 'Raiden Ei', age: 27, city: 'Inazuma', active: true },
  { id: 4, name: 'Nahida', age: 11, city: 'Sumeru', active: true },
  { id: 5, name: 'Furina', age: 19, city: 'Fontaine', active: false },
  { id: 5, name: 'Mavuika', age: 30, city: 'Natlan', active: true }
]
```

D. Use the same array and demonstrate a sample code using the unshift() method on the array.



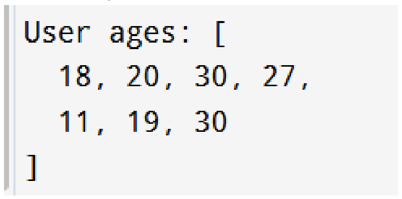
```
After unshift: [
  { id: 0, name: 'Columbina', age: 18, city: 'Nodkrai', active: false },
  { id: 1, name: 'Venti', age: 20, city: 'Mondstadt', active: true },
  { id: 2, name: 'Zhongli', age: 30, city: 'Liyue', active: false },
  { id: 3, name: 'Raiden Ei', age: 27, city: 'Inazuma', active: true },
  { id: 4, name: 'Nahida', age: 11, city: 'Sumeru', active: true },
  { id: 5, name: 'Furina', age: 19, city: 'Fontaine', active: false },
  { id: 5, name: 'Mavuika', age: 30, city: 'Natlan', active: true }
]
```

E. Use the same array and demonstrate a sample code using the filter() method on the array.



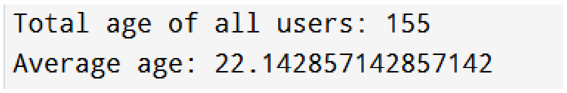
```
Active users: [
  { id: 1, name: 'Venti', age: 20, city: 'Mondstadt', active: true },
  { id: 3, name: 'Raiden Ei', age: 27, city: 'Inazuma', active: true },
  { id: 4, name: 'Nahida', age: 11, city: 'Sumeru', active: true },
  { id: 5, name: 'Mavuika', age: 30, city: 'Natlan', active: true }
]
Users over 20: [
  { id: 2, name: 'Zhongli', age: 30, city: 'Liyue', active: false },
  { id: 3, name: 'Raiden Ei', age: 27, city: 'Inazuma', active: true },
  { id: 5, name: 'Mavuika', age: 30, city: 'Natlan', active: true }
]
```

F. Use the same array and demonstrate a sample code using the map() method on the array.



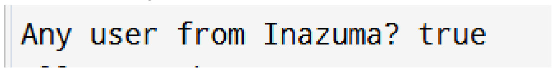
```
User ages: [
  18, 20, 30, 27,
  11, 19, 30
]
```

G. Use the same array and demonstrate a sample code using the reduce() method on the array.



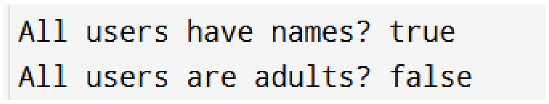
```
Total age of all users: 155
Average age: 22.142857142857142
```

H. Use the same array and demonstrate a sample code using the some() method on the array.



```
Any user from Inazuma? true
```

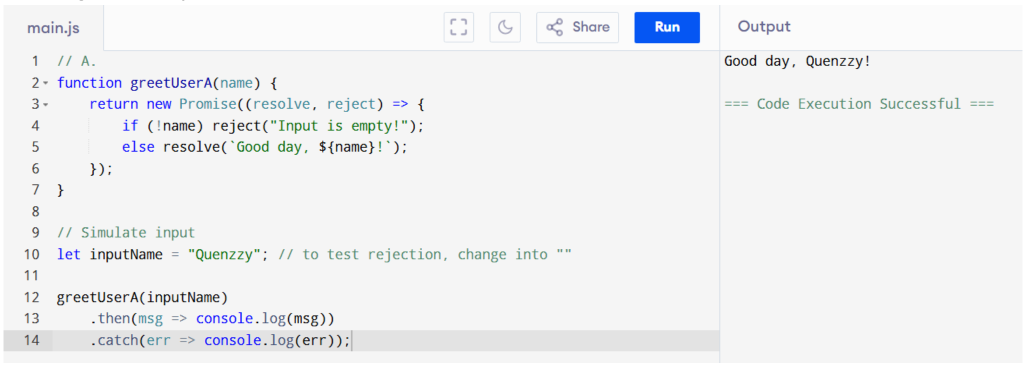
I. Use the same array and demonstrate a sample code using the every() method on the array.



```
All users have names? true
All users are adults? false
```

II. ASYNCHRONOUS PROGRAMMING:

A. We first have an input field asking for the user's name. Create a Promise that rejects if that input field is empty, and resolves with the input, greeting the user with "good day, <name of user here>!" on the DOM



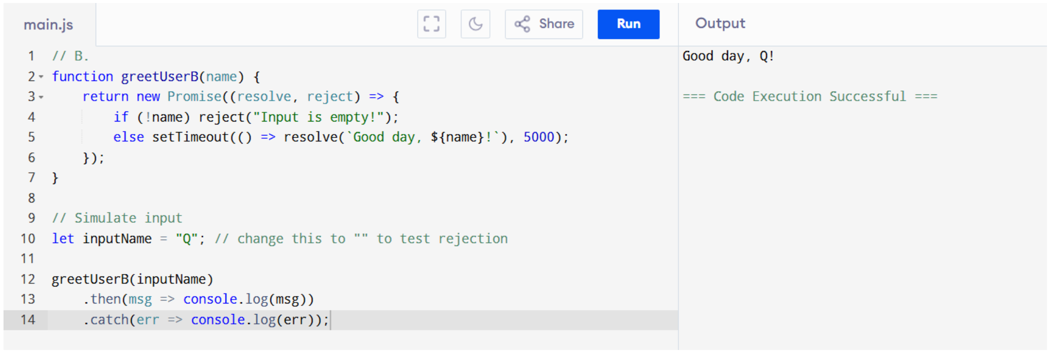
main.js

```
// A.
function greetUserA(name) {
    return new Promise((resolve, reject) => {
        if (!name) reject("Input is empty!");
        else resolve(`Good day, ${name}!`);
    });
}

// Simulate input
let inputName = "Quenzzy"; // to test rejection, change into ""

greetUserA(inputName)
    .then(msg => console.log(msg))
    .catch(err => console.log(err));
```

Output

```
Good day, Quenzzy!

=== Code Execution Successful ===
```

B. We first have an input field asking for the user's name. Create a Promise that rejects if that input field is empty, and resolves **after 5 seconds** with the input, greeting the user with "good day, <name of user here>!" on the DOM.

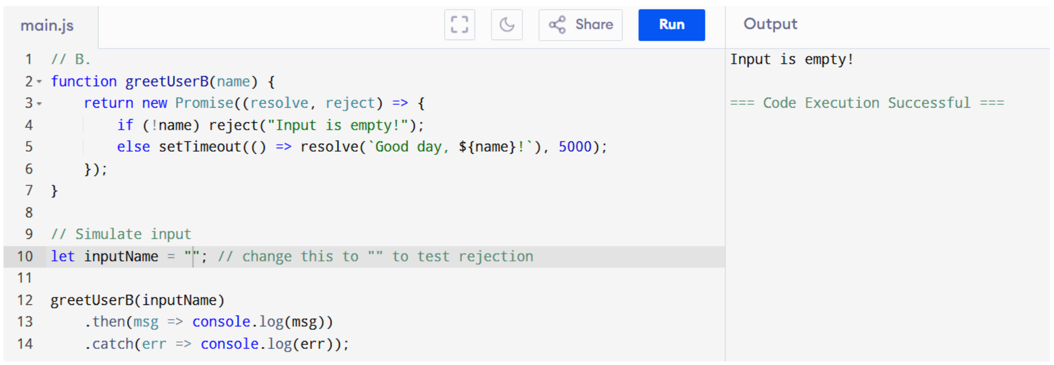


main.js

```
// B.
function greetUserB(name) {
    return new Promise((resolve, reject) => {
        if (!name) reject("Input is empty!");
        else setTimeout(() => resolve(`Good day, ${name}!`), 5000);
    });
}

// Simulate input
let inputName = "Q"; // change this to "" to test rejection

greetUserB(inputName)
    .then(msg => console.log(msg))
    .catch(err => console.log(err));
```

Output

```
Good day, Q!

=== Code Execution Successful ===
```



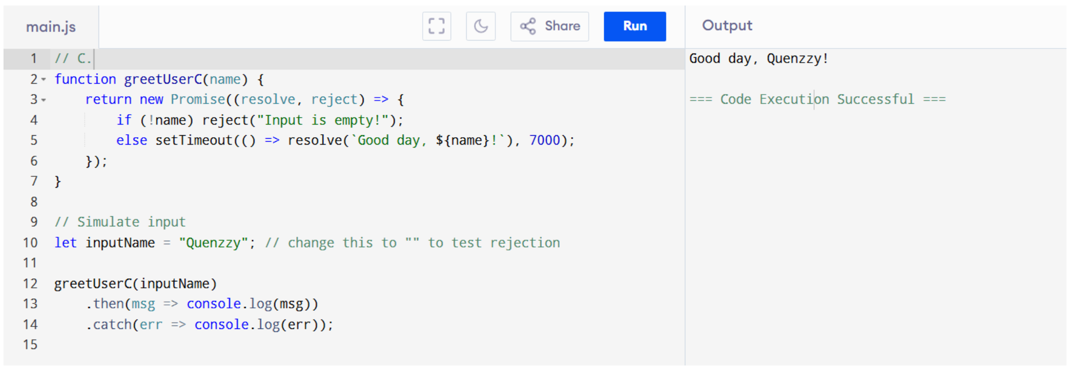
main.js

```
// B.
function greetUserB(name) {
    return new Promise((resolve, reject) => {
        if (!name) reject("Input is empty!");
        else setTimeout(() => resolve(`Good day, ${name}!`), 5000);
    });
}

// Simulate input
let inputName = ""; // change this to "" to test rejection

greetUserB(inputName)
    .then(msg => console.log(msg))
    .catch(err => console.log(err));
```

Output

```
Input is empty!

=== Code Execution Successful ===
```

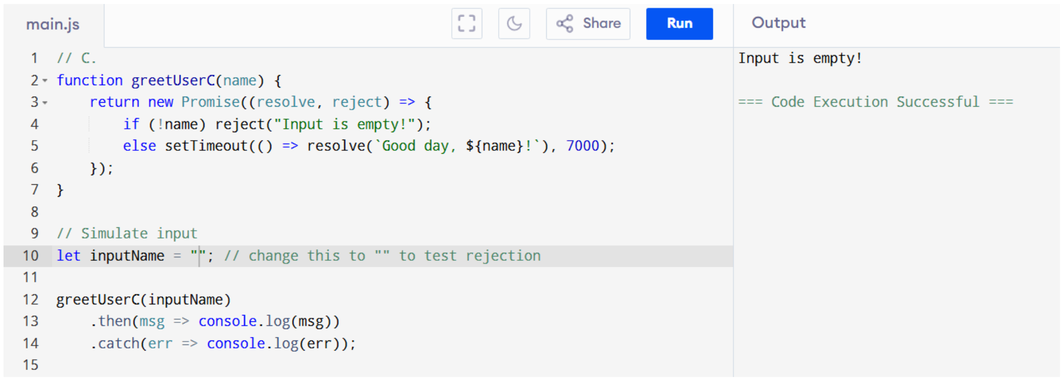
C. We first have an input field asking for the user's name. Create a Promise that rejects if that input field is empty, and resolves **after 7 seconds** with the input, greeting the user with "good day, <name of user here>!" on the DOM.


main.js

```
// C.
function greetUserC(name) {
    return new Promise((resolve, reject) => {
        if (!name) reject("Input is empty!");
        else setTimeout(() => resolve(`Good day, ${name}!`), 7000);
    });
}

// Simulate input
let inputName = "Quenzzy"; // change this to "" to test rejection

greetUserC(inputName)
    .then(msg => console.log(msg))
    .catch(err => console.log(err));
```

Output

```
Good day, Quenzzy!

=== Code Execution Successful ===
```

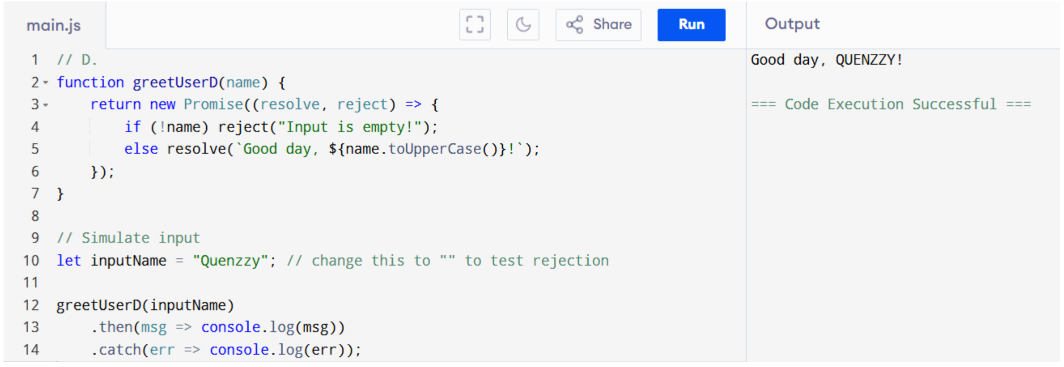

main.js

```
// C.
function greetUserC(name) {
    return new Promise((resolve, reject) => {
        if (!name) reject("Input is empty!");
        else setTimeout(() => resolve(`Good day, ${name}!`), 7000);
    });
}

// Simulate input
let inputName = ""; // change this to "" to test rejection

greetUserC(inputName)
    .then(msg => console.log(msg))
    .catch(err => console.log(err));
```

Output

```
Input is empty!

=== Code Execution Successful ===
```

D. We first have an input field asking for the user's name. Create a Promise that rejects if that input field is empty, and resolves with the input being in **uppercase format**, greeting the user with "good day, <name of user here>!" on the DOM.

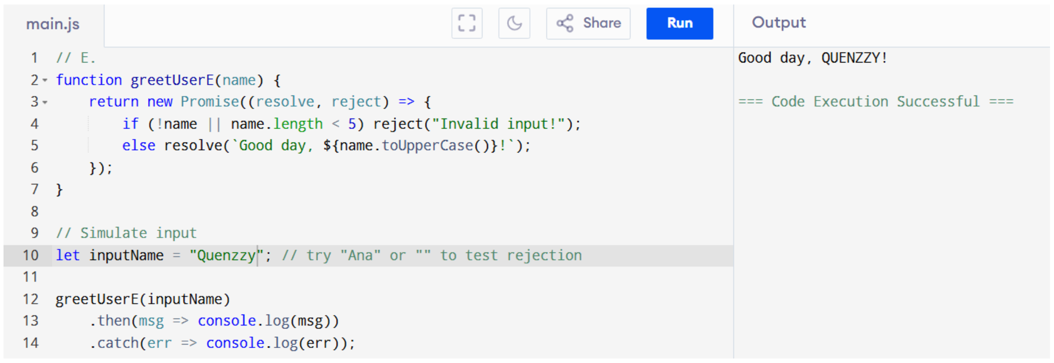

main.js

```
// D.
function greetUserD(name) {
    return new Promise((resolve, reject) => {
        if (!name) reject("Input is empty!");
        else resolve(`Good day, ${name.toUpperCase()}!`);
    });
}

// Simulate input
let inputName = "Quenzzy"; // change this to "" to test rejection

greetUserD(inputName)
    .then(msg => console.log(msg))
    .catch(err => console.log(err));
```

Output

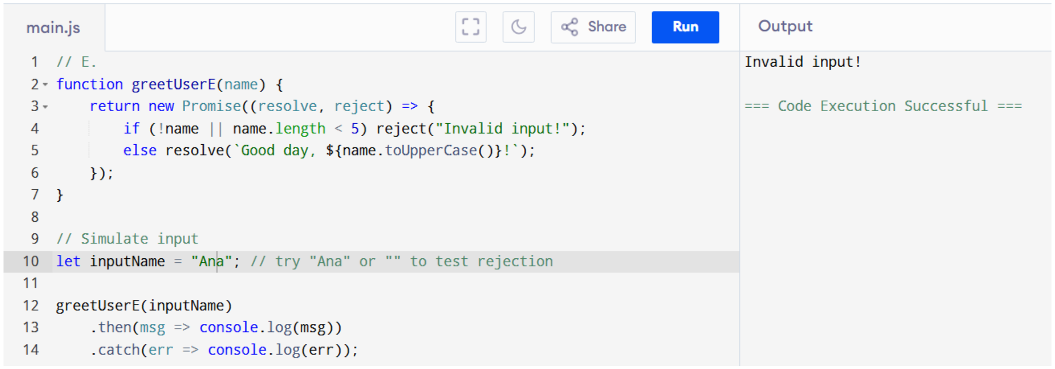
```
Good day, QUENZZY!

=== Code Execution Successful ===
```

E. We first have an input field asking for the user's name. Create a Promise that rejects if that input field is empty and is **less than five characters**, and resolves with the input being in **uppercase format**, greeting the user with "good day, <name of user here>!" on the DOM.



```javascript
// E.
function greetUserE(name) {
    return new Promise((resolve, reject) => {
        if (!name || name.length < 5) reject("Invalid input!");
        else resolve(`Good day, ${name.toUpperCase()}!`);
    });
}

// Simulate input
let inputName = "Quenzzy"; // try "Ana" or "" to test rejection

greetUserE(inputName)
    .then(msg => console.log(msg))
    .catch(err => console.log(err));
```

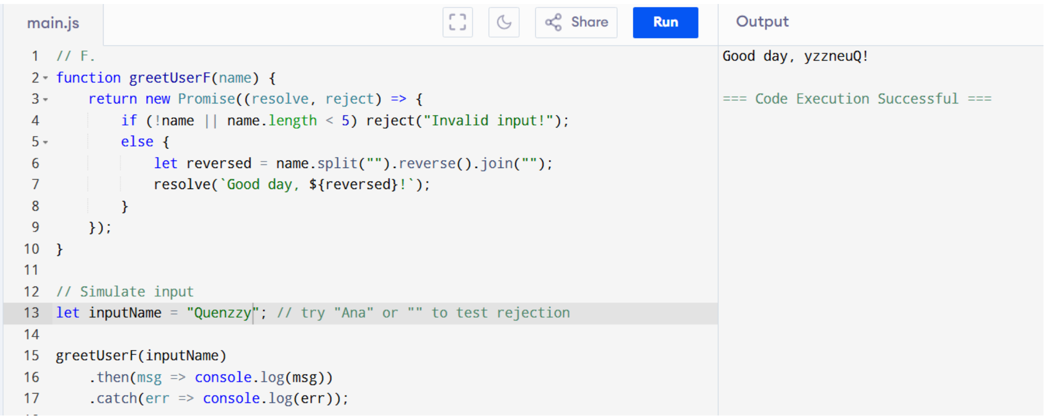
```
Good day, QUENZZY!

=== Code Execution Successful ===
```



```javascript
// E.
function greetUserE(name) {
    return new Promise((resolve, reject) => {
        if (!name || name.length < 5) reject("Invalid input!");
        else resolve(`Good day, ${name.toUpperCase()}!`);
    });
}

// Simulate input
let inputName = "Ana"; // try "Ana" or "" to test rejection

greetUserE(inputName)
    .then(msg => console.log(msg))
    .catch(err => console.log(err));
```

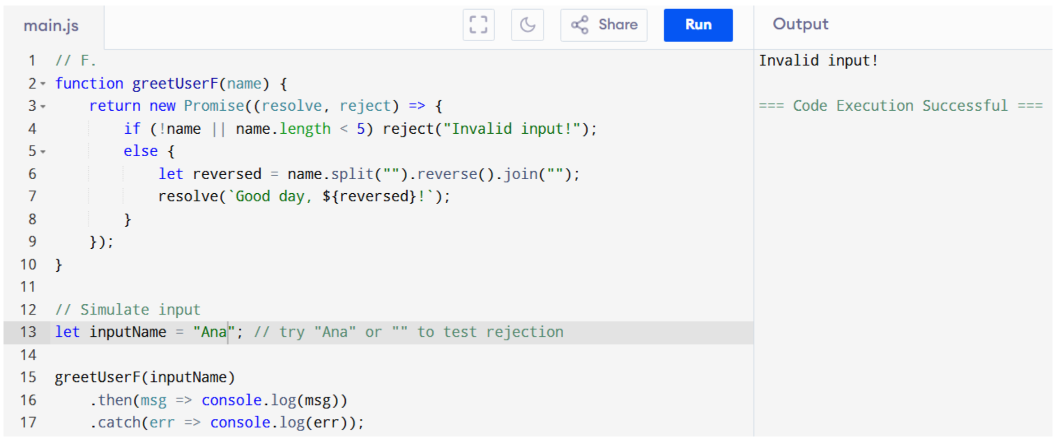
```
Invalid input!

=== Code Execution Successful ===
```

F. We first have an input field asking for the user's name. Create a Promise that rejects if that input field is empty and is **less than five characters**, and resolves with the input being in **reversed format**, greeting the user with "good day, <name of user here>!" on the DOM.



```javascript
// F.
function greetUserF(name) {
    return new Promise((resolve, reject) => {
        if (!name || name.length < 5) reject("Invalid input!");
        else {
            let reversed = name.split("").reverse().join("");
            resolve(`Good day, ${reversed}!`);
        }
    });
}

// Simulate input
let inputName = "Quenzzy"; // try "Ana" or "" to test rejection

greetUserF(inputName)
    .then(msg => console.log(msg))
    .catch(err => console.log(err));
```

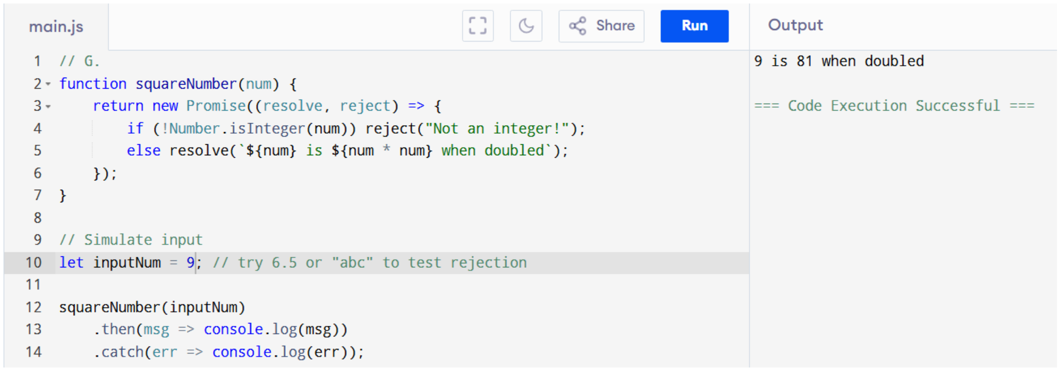
```
Good day, yzzneuQ!

=== Code Execution Successful ===
```



```javascript
// F.
function greetUserF(name) {
    return new Promise((resolve, reject) => {
        if (!name || name.length < 5) reject("Invalid input!");
        else {
            let reversed = name.split("").reverse().join("");
            resolve(`Good day, ${reversed}!`);
        }
    });
}

// Simulate input
let inputName = "Ana"; // try "Ana" or "" to test rejection

greetUserF(inputName)
    .then(msg => console.log(msg))
    .catch(err => console.log(err));
```

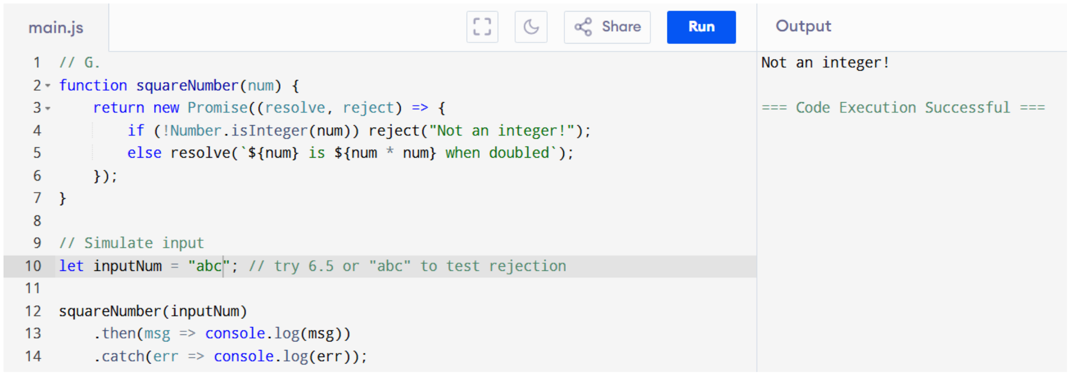
```
Invalid input!

=== Code Execution Successful ===
```

G. We first have an input field asking the user to input a number. Create a Promise that rejects if the inputted value is not an integer and resolves with the integer input being **squared**, printing the string “<number inputted> is <squared number> when doubled ” on the DOM.



```javascript
// G.
function squareNumber(num) {
    return new Promise((resolve, reject) => {
        if (!Number.isInteger(num)) reject("Not an integer!");
        else resolve(`${num} is ${num * num} when doubled`);
    });
}

// Simulate input
let inputNum = 9; // try 6.5 or "abc" to test rejection

squareNumber(inputNum)
    .then(msg => console.log(msg))
    .catch(err => console.log(err));
```

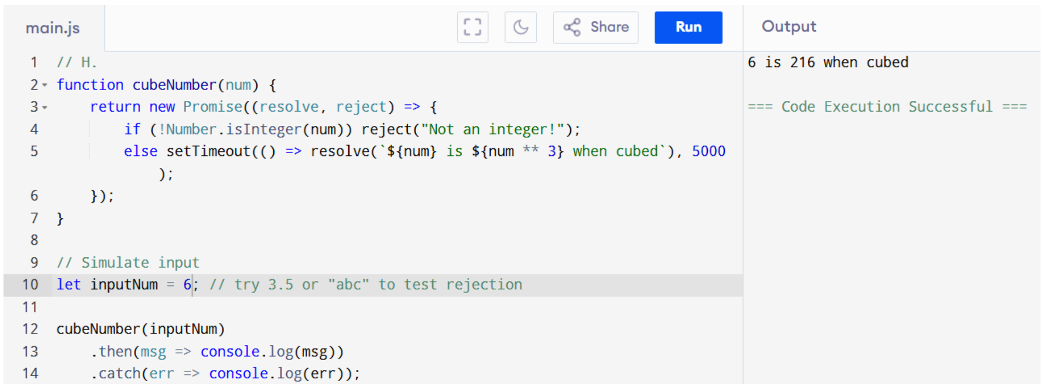
```
9 is 81 when doubled

=== Code Execution Successful ===
```



```javascript
// G.
function squareNumber(num) {
    return new Promise((resolve, reject) => {
        if (!Number.isInteger(num)) reject("Not an integer!");
        else resolve(`${num} is ${num * num} when doubled`);
    });
}

// Simulate input
let inputNum = "abc"; // try 6.5 or "abc" to test rejection

squareNumber(inputNum)
    .then(msg => console.log(msg))
    .catch(err => console.log(err));
```

```
Not an integer!

=== Code Execution Successful ===
```

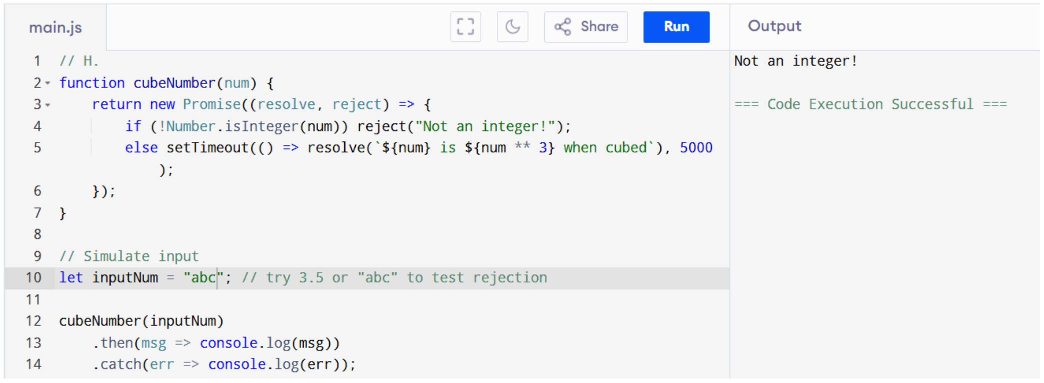
H. We first have an input field asking the user to input a number. Create a Promise that rejects if the inputted value is not an integer and resolves after **5 seconds** with the integer input being **cubed**, printing the string “<number inputted> is <cubed number> when cubed” on the DOM.



main.js Share Run Output

```javascript
// H.
function cubeNumber(num) {
    return new Promise((resolve, reject) => {
        if (!Number.isInteger(num)) reject("Not an integer!");
        else setTimeout(() => resolve(`${num} is ${num ** 3} when cubed`), 5000
            );
    });
}

// Simulate input
let inputNum = 6; // try 3.5 or "abc" to test rejection

cubeNumber(inputNum)
    .then(msg => console.log(msg))
    .catch(err => console.log(err));
```

```
6 is 216 when cubed

=== Code Execution Successful ===
```

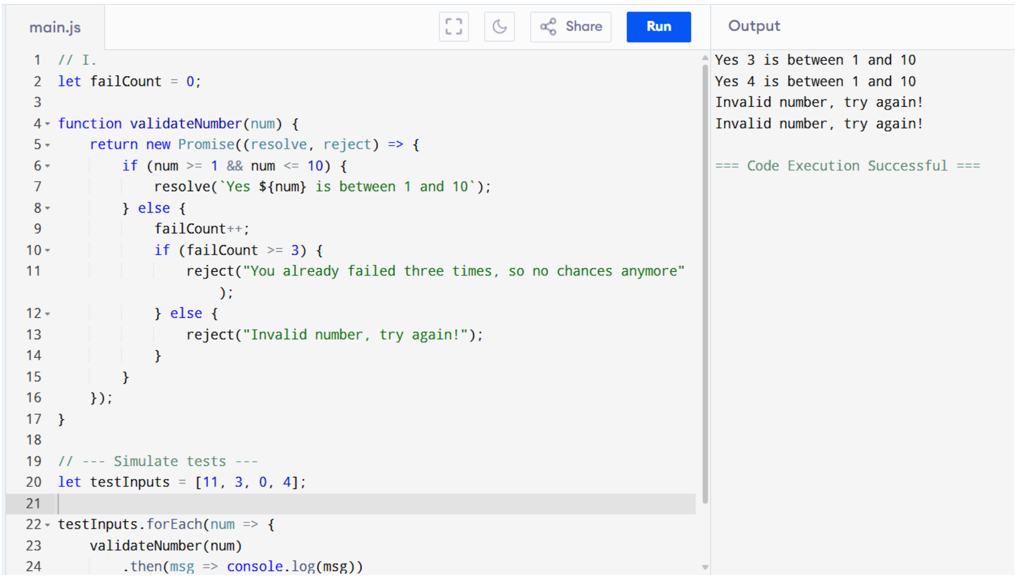


main.js Share Run Output

```javascript
// H.
function cubeNumber(num) {
    return new Promise((resolve, reject) => {
        if (!Number.isInteger(num)) reject("Not an integer!");
        else setTimeout(() => resolve(`${num} is ${num ** 3} when cubed`), 5000
            );
    });
}

// Simulate input
let inputNum = "abc"; // try 3.5 or "abc" to test rejection

cubeNumber(inputNum)
    .then(msg => console.log(msg))
    .catch(err => console.log(err));
```

```
Not an integer!

=== Code Execution Successful ===
```

I. We first have an input field asking the user to input a number. Create a Promise that rejects if the inputted value is not **between 1 and 10** and resolves by printing the string "Yes <number inputted> is between 1 and 10" on the DOM. If the user fails three times to give a number between 1 and 10, we'll just print "You already failed three times, so no chances anymore".



main.js Share Run Output

```javascript
// I.
let failCount = 0;

function validateNumber(num) {
    return new Promise((resolve, reject) => {
        if (num >= 1 && num <= 10) {
            resolve(`Yes ${num} is between 1 and 10`);
        } else {
            failCount++;
            if (failCount >= 3) {
                reject("You already failed three times, so no chances anymore"
                    );
            } else {
                reject("Invalid number, try again!");
            }
        }
    });
}

// --- Simulate tests ---
let testInputs = [11, 3, 0, 4];

testInputs.forEach(num => {
    validateNumber(num)
        .then(msg => console.log(msg))
```

```
Yes 3 is between 1 and 10
Yes 4 is between 1 and 10
Invalid number, try again!
Invalid number, try again!

=== Code Execution Successful ===
```

III. GITHUB LINK

isayasi/1_JAVASCRIPT_OBJECTS_ASYNC_PROGRAMMING