Quiz 3

Student ID: _____ Student Name: _____

1. [5 points] Attach a event handler to “Get Users” button in the HTML which has the features listed below:
   1.1. make a request to url: https://randomuser.me/api. The returned JSON format looks like below. Here 3 users are just an example, it may return more users.


```
{
  "results": [
    {
      "gender": "female",
      "name": {
        "title": "Miss",
        "first": "Ethel",
        "last": "Hawkins"
      }
    },
    {
      "gender": "female",
      "name": {
        "title": "Miss",
        "first": "Lily",
        "last": "Webb"
      }
    },
    {
      "gender": "female",
      "name": {
        "title": "Ms",
        "first": "Begüm",
        "last": "Özbir"
      }
    }
}
```


   1.2. Retrieve user’s filter condition gender from the user's input in the browser, then print all filtered users’ first name and last name into console. – just print to console, no need to insert into HTML.

1.3. Here is the HTML code. You're not allowed to make any change. The code is complete.

```
<!DOCTYPE html>
<html lang="en-US">

<head>
    <title>Prog 1</title>
    <script src="index.js"></script>
</head>

<body>
    <input type="text" name="gender" id="filterCondition" />
    <button id="btn">Get Users</button>
</body>

</html>
```

1.4. Write JS code to implement the feature in the index.js file which is linked in above HTML file

```
window.onload = function() {
    document.getElementById("btn").onclick = async function() {

        const gender = document.getElementById('filterCondition').value;

        let responseBody = await fetch('https://randomuser.me/api?results=10');
        let userArr = await responseBody.json();
        userArr.results.filter(user => user.gender === gender)
            .forEach(user => {
                console.log(user.name.first + ' ' + user.name.last);
            });
    }
}
```

2. [5 points] Redo the same feature in previous question using RxJS Observable.
   2.1. Assume Rxjs library is added in HTML <head>:

   <script src="https://cdnjs.cloudflare.com/ajax/libs/rxjs/6.6.7/rxjs.umd.js"></script>

   2.2. The filter user feature MUST be done using Observable Operators. No credit if using Array filter() method.
   2.3. All necessary operators are imported:

```
const { Observable, from } = rxjs;
const { map, filter, reduce } = rxjs.operators;
```

```
const { from } = rxjs;
const { map, filter } = rxjs.operators;

window.onload = function() {
    document.getElementById("btn").onclick = async function() {

        const gender = document.getElementById('filterCondition').value;

        let result = await fetch('https://randomuser.me/api/?results=10').then(re
sponse => response.json()).then(json => json.results);
        from(result)
            .pipe(
                filter(user => user.gender === gender)
            )
            .subscribe(user => user.name.first + ' ' + user.name.last);
    }
}
```

3. [5 points] Write an implementation for the Observer Pattern where observers have the following format: `{'event': [observers]}`
   For example:

```
{
   'eat': [function1, function2],
   'study': [function3, function4, function5]
}
```

This Observable/Subject should be used as following:

```
const subject = new Subject();
subject.on('eat', console.log);
subject.on('study', console.log);

function foo(msg) {
    console.log('foo: ' + msg);
}
subject.on('eat', foo);
subject.on('study', foo);

subject.emit('eat', 'Corn');
subject.emit('study', 'cs445');
//output
// Corn
// foo: Corn
// cs445
// foo: cs445
subject.off('eat', foo);
subject.emit('eat', 'Banana');
//output
//Banana
```

Your Solution:

```
class Subject {
    observers = {};

    on(event, fn) {
        if (!this.observers[event]) {
            this.observers[event] = [fn];
        } else {
            this.observers[event].push(fn);
        }
    }

    emit(event, message) {
        if (this.observers[event]) {
            this.observers[event].forEach(fn => fn(message
));
        }
    }

    off(event, fn) {
        this.observers[event] = this.observers[event].filt
er(f => f !== fn);
    }

}
```

[5 points] The `fibonacci` function is used to calculate the `Fibonacci` sequence.

```
function fibonacci(n) {
    if (n <= 1) {
        return 1
    }
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

The problem of the `fibonacci` function is there are so many redundant calculation.

Refactor the `fibonacci` function using Memoization technique. For the same input to the function, only the first time do the calculation, the remaining method call with the same input grabs the result from cache. The cache isn't allowed to leak to global environment.

```
const fibonacci = (function() {
    const memo = {};

    function f(n) {
        let value;

        if (n in memo) {
            value = memo[n];
        } else {
            if (n === 0 || n === 1)
                value = n;
            else
                value = f(n - 1) + f(n - 2);
            memo[n] = value;
        }
        return value;
    }
    return f;
})();
```

```
const fibonacci = (function() {
    const memo = new Map();

    function f(n) {
        let value = memo.get(n);
        if (!value) {
            if (n === 0 || n === 1)
                value = n;
```

```
            else
                value = f(n - 1) + f(n - 2);
            memo.set(n, value);
        }
        return value;
    }
    return f;
})();
```