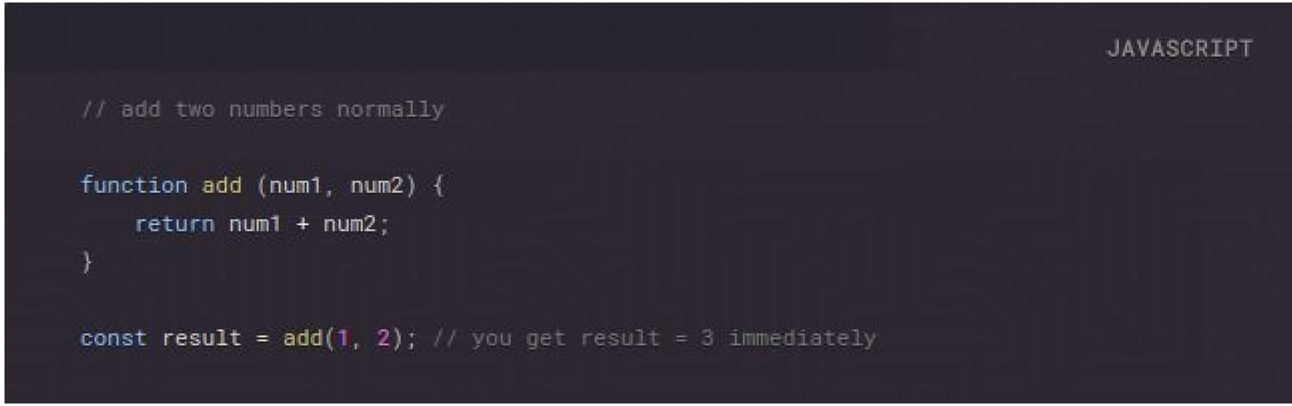
Promises (= Monadic Futures)

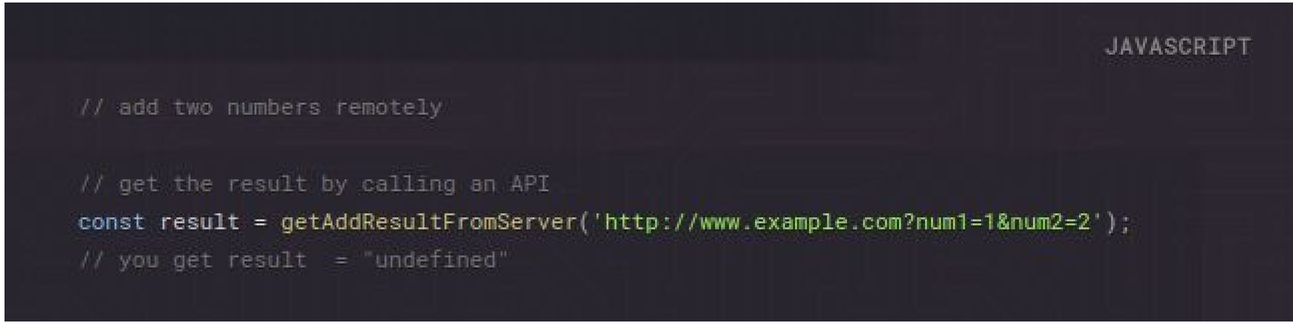
PK, SM

Synchronous vs Async



JAVASCRIPT

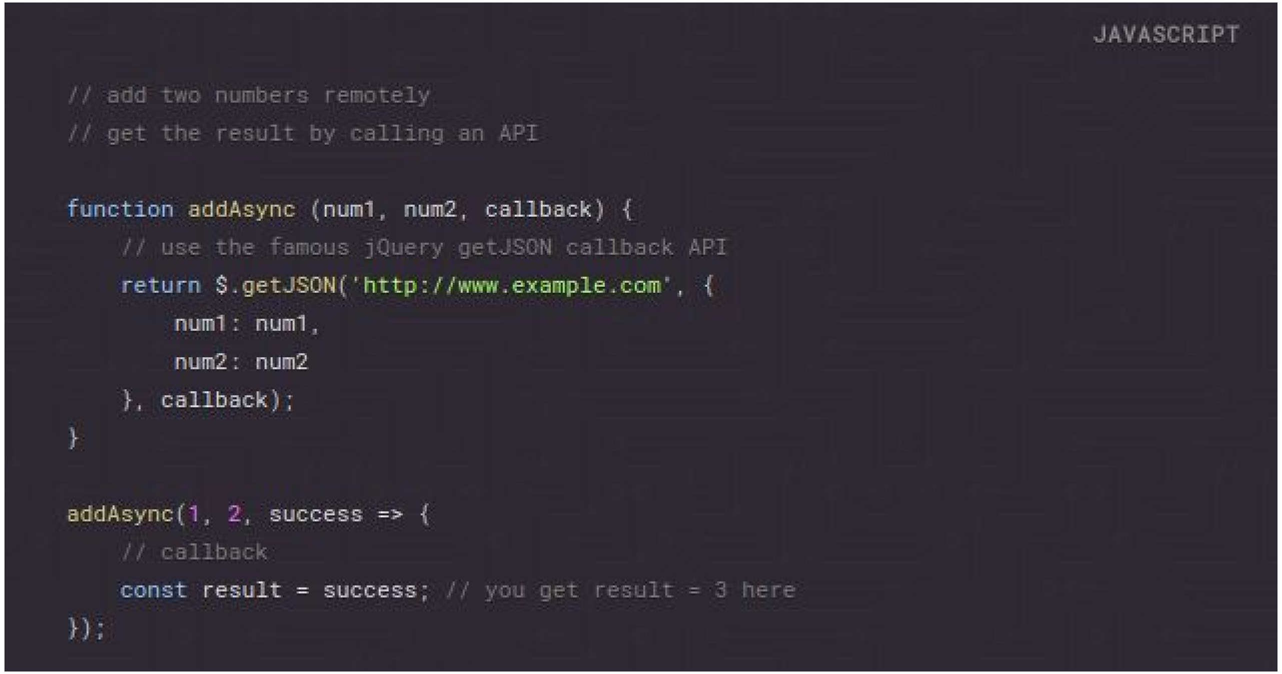
```javascript
// add two numbers normally

function add (num1, num2) {
    return num1 + num2;
}

const result = add(1, 2); // you get result = 3 immediately
```



JAVASCRIPT

```javascript
// add two numbers remotely

// get the result by calling an API
const result = getAddResultFromServer('http://www.example.com?num1=1&num2=2');
// you get result  = "undefined"
```

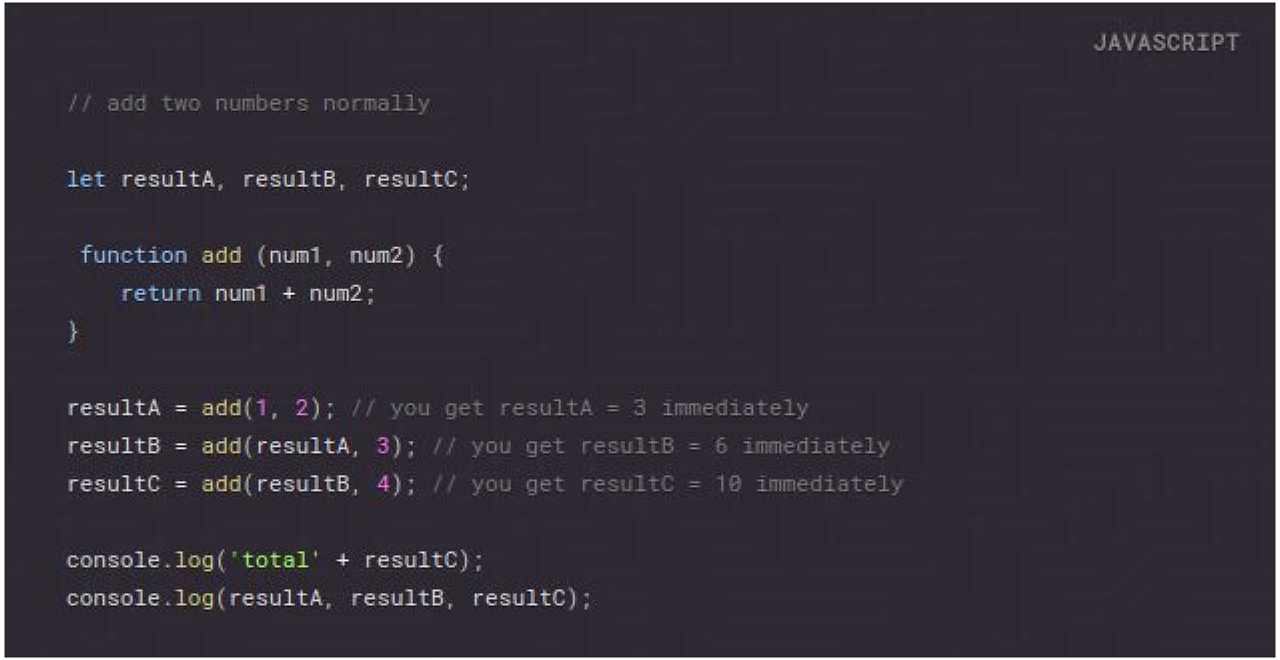
Before Promises (using callbacks)



JAVASCRIPT

```javascript
// add two numbers remotely
// get the result by calling an API

function addAsync (num1, num2, callback) {
    // use the famous jQuery getJSON callback API
    return $.getJSON('http://www.example.com', {
        num1: num1,
        num2: num2
    }, callback);
}

addAsync(1, 2, success => {
    // callback
    const result = success; // you get result = 3 here
});
```

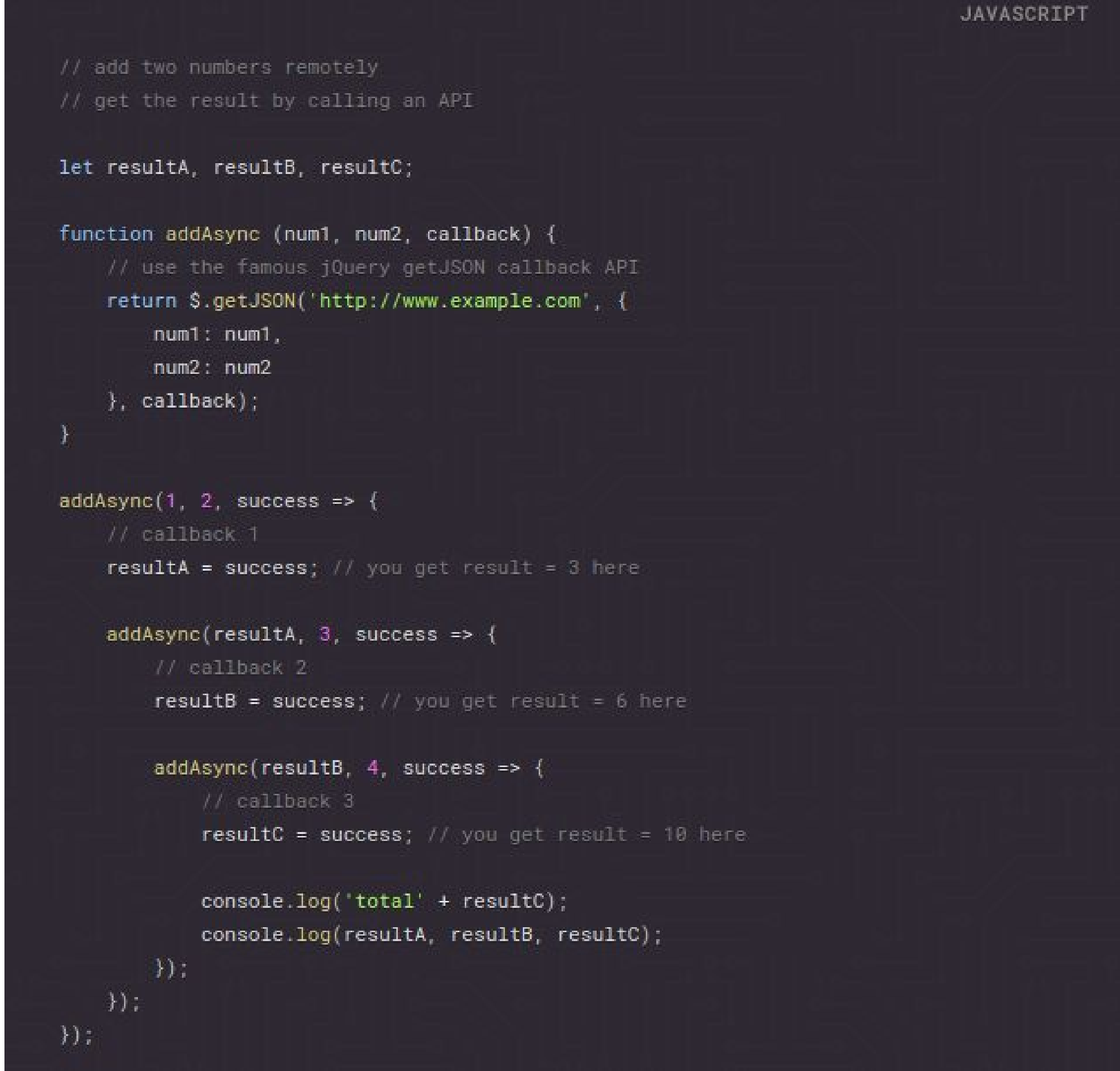
Subsequent Actions



JAVASCRIPT

```javascript
// add two numbers normally

let resultA, resultB, resultC;

 function add (num1, num2) {
    return num1 + num2;
}

resultA = add(1, 2); // you get resultA = 3 immediately
resultB = add(resultA, 3); // you get resultB = 6 immediately
resultC = add(resultB, 4); // you get resultC = 10 immediately

console.log('total' + resultC);
console.log(resultA, resultB, resultC);
```

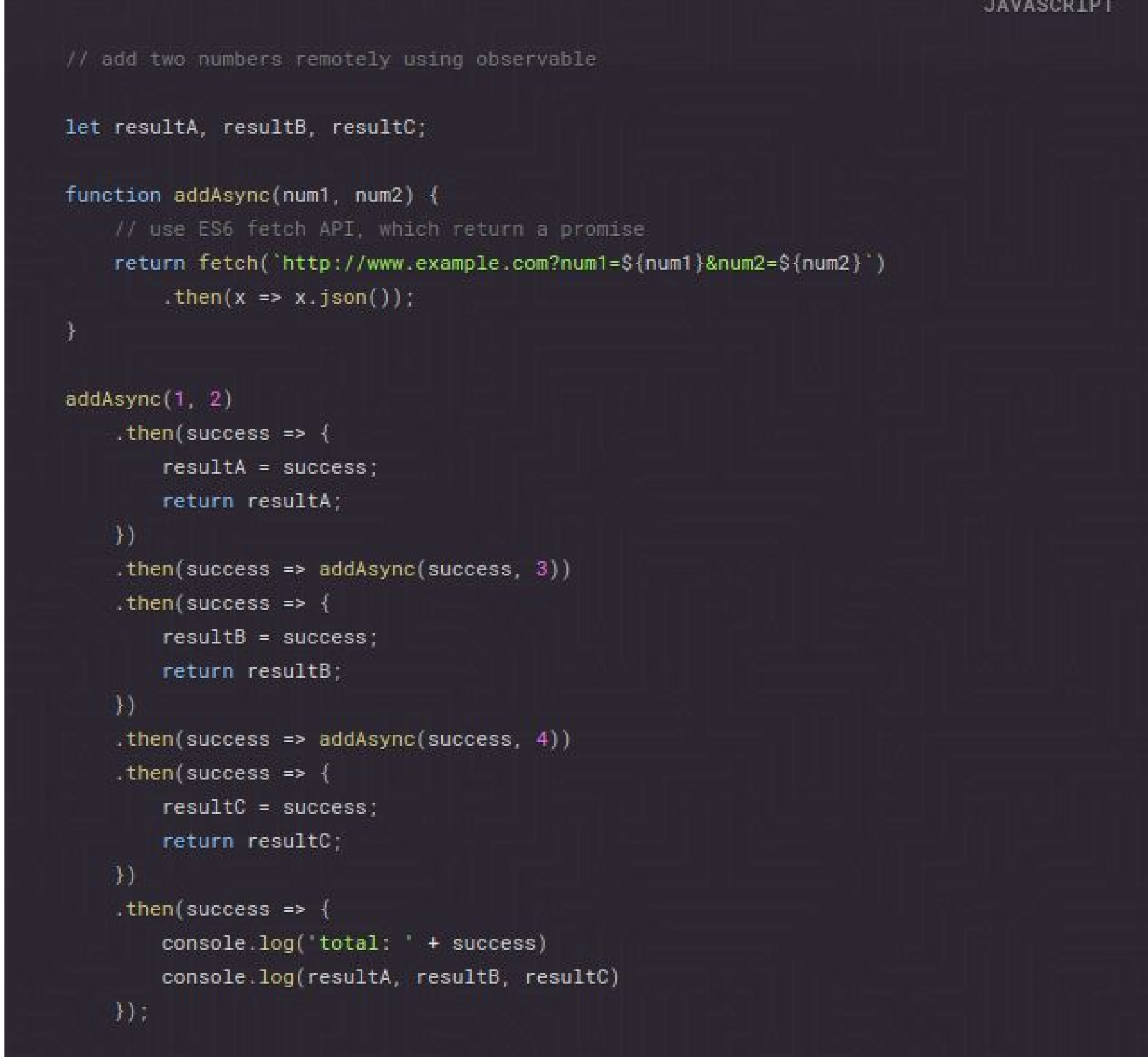
Using multiple actions (callback hell)



```javascript
// add two numbers remotely
// get the result by calling an API

let resultA, resultB, resultC;

function addAsync (num1, num2, callback) {
    // use the famous jQuery getJSON callback API
    return $.getJSON('http://www.example.com', {
        num1: num1,
        num2: num2
    }, callback);
}

addAsync(1, 2, success => {
    // callback 1
    resultA = success; // you get result = 3 here

    addAsync(resultA, 3, success => {
        // callback 2
        resultB = success; // you get result = 6 here

        addAsync(resultB, 4, success => {
            // callback 3
            resultC = success; // you get result = 10 here

            console.log('total' + resultC);
            console.log(resultA, resultB, resultC);
        });
    });
});
```

Using Promises

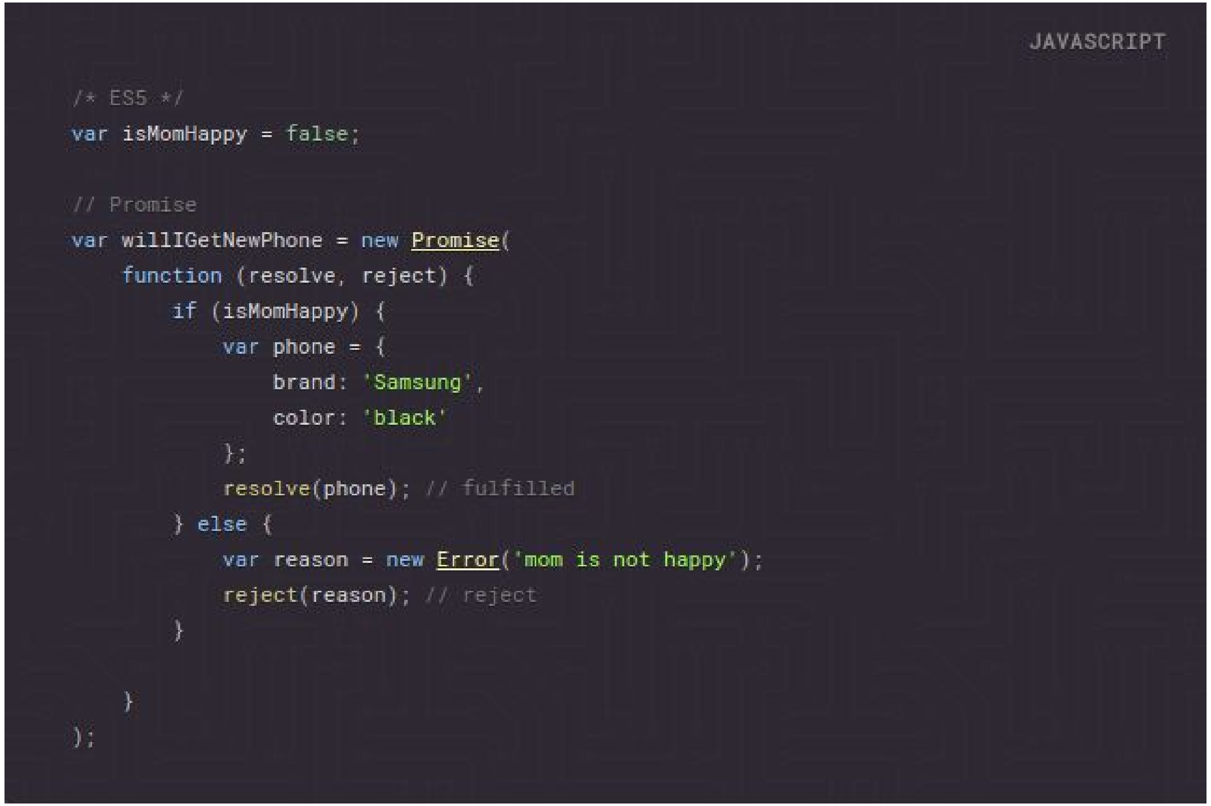
- Will be repeated at the end.
- Escape from callback hell.



```javascript
// add two numbers remotely using observable

let resultA, resultB, resultC;

function addAsync(num1, num2) {
    // use ES6 fetch API, which return a promise
    return fetch(`http://www.example.com?num1=${num1}&num2=${num2}`)
        .then(x => x.json());
}

addAsync(1, 2)
    .then(success => {
        resultA = success;
        return resultA;
    })
    .then(success => addAsync(success, 3))
    .then(success => {
        resultB = success;
        return resultB;
    })
    .then(success => addAsync(success, 4))
    .then(success => {
        resultC = success;
        return resultC;
    })
    .then(success => {
        console.log('total: ' + success)
        console.log(resultA, resultB, resultC)
    });
```

Promises

- 3 states
  - Pending
  - Resolved
  - Rejected
- Special functions:
  - resolve()
  - reject()

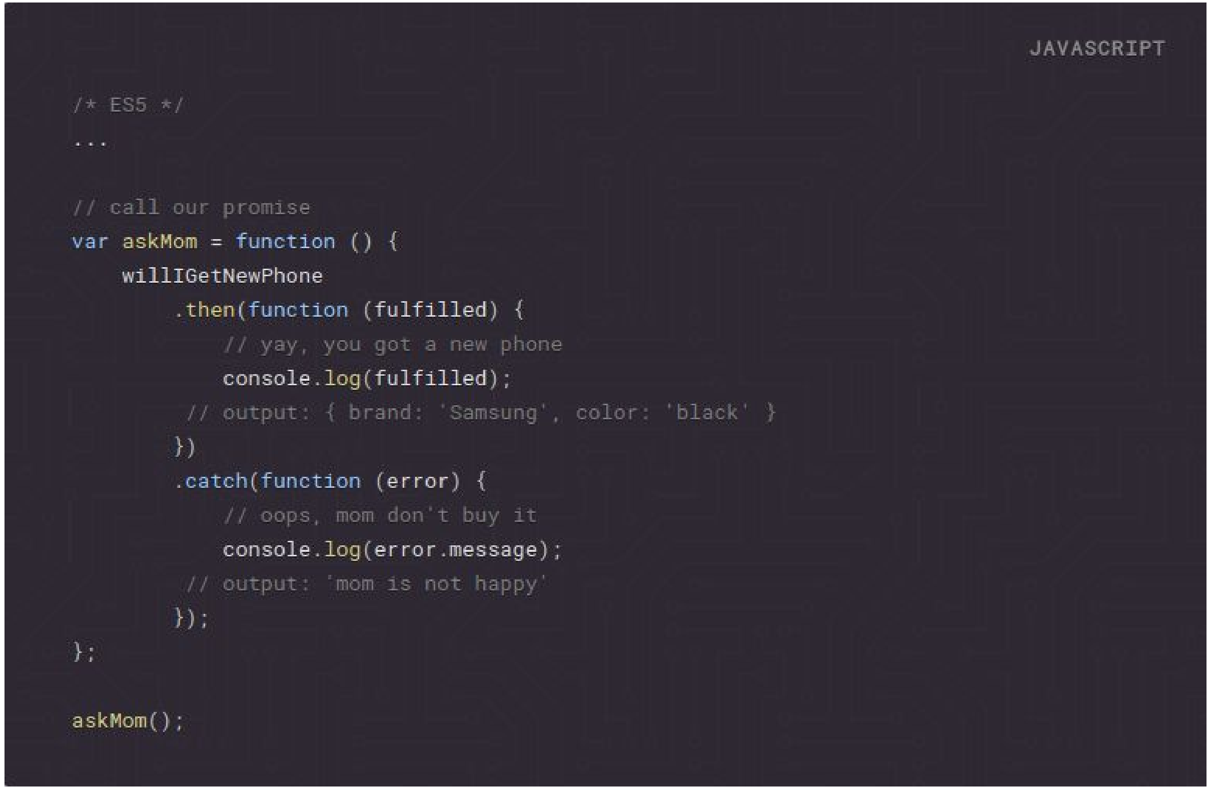


JAVASCRIPT

```
/* ES5 */
var isMomHappy = false;

// Promise
var willIGetNewPhone = new Promise(
    function (resolve, reject) {
        if (isMomHappy) {
            var phone = {
                brand: 'Samsung',
                color: 'black'
            };
            resolve(phone); // fulfilled
        } else {
            var reason = new Error('mom is not happy');
            reject(reason); // reject
        }

    }
);
```

Consuming Promises

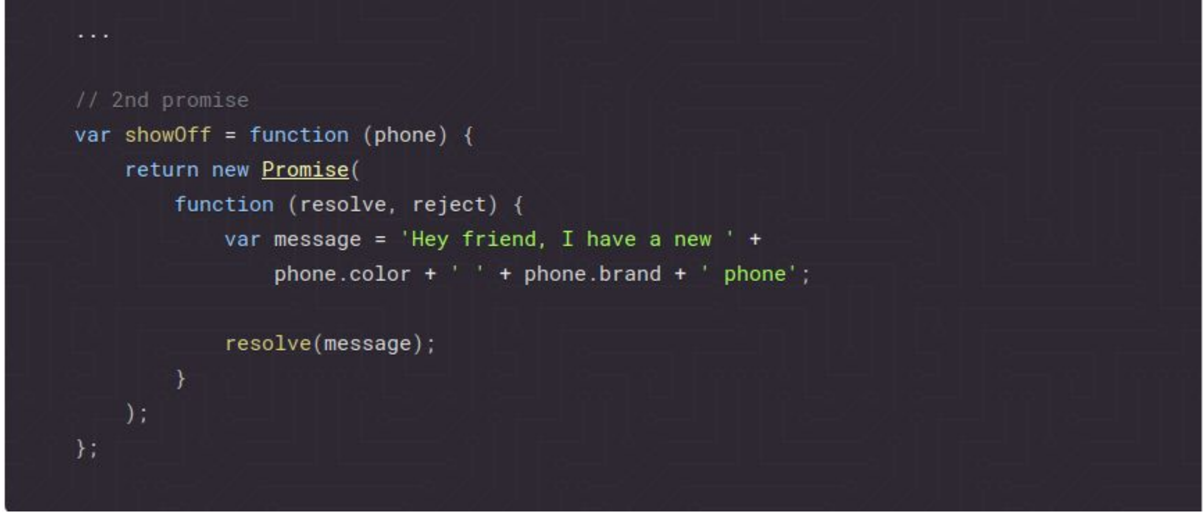
- Special functions
  - then()
  - catch()
- Makes it monadic



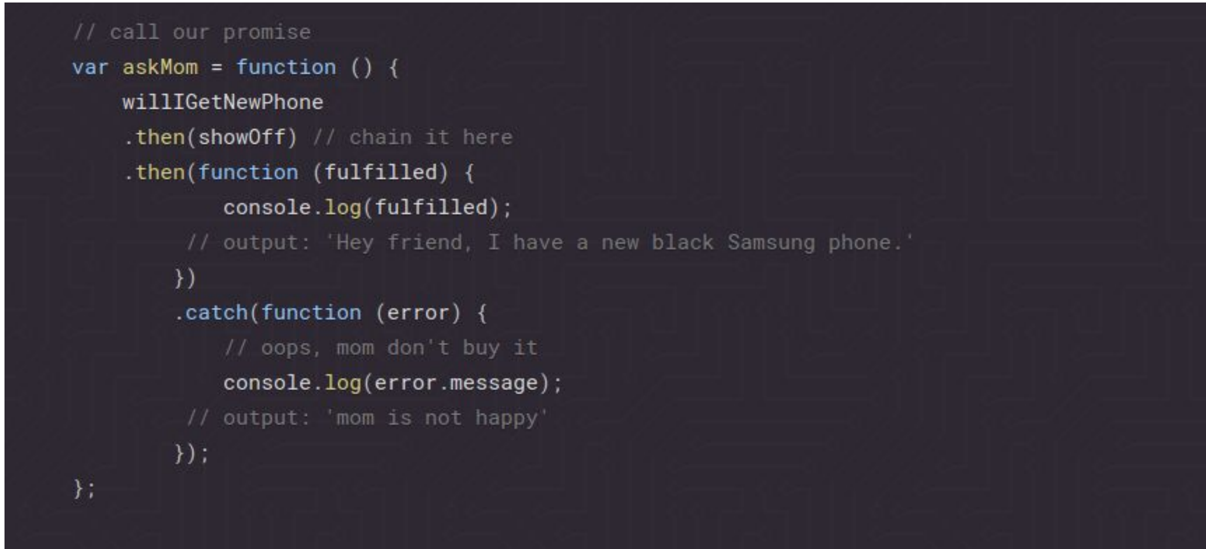
JAVASCRIPT

```
/* ES5 */
...

// call our promise
var askMom = function () {
    willIGetNewPhone
        .then(function (fulfilled) {
            // yay, you got a new phone
            console.log(fulfilled);
         // output: { brand: 'Samsung', color: 'black' }
        })
        .catch(function (error) {
            // oops, mom don't buy it
            console.log(error.message);
         // output: 'mom is not happy'
        });
};

askMom();
```

Chaining Promises



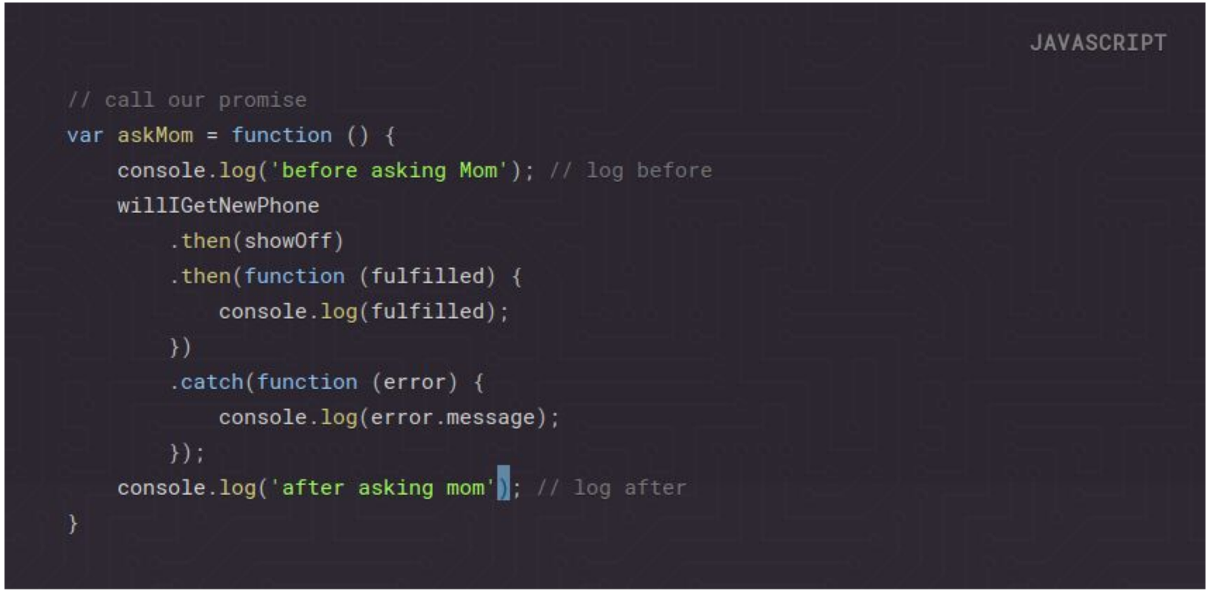
```
...

// 2nd promise
var showOff = function (phone) {
    return new Promise(
        function (resolve, reject) {
            var message = 'Hey friend, I have a new ' +
                phone.color + ' ' + phone.brand + ' phone';

            resolve(message);
        }
    );
};
```



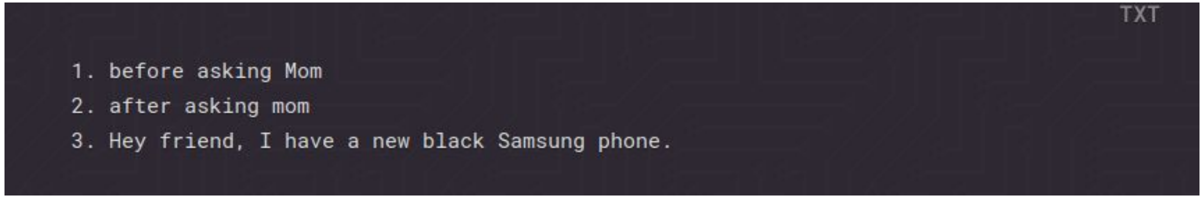
```
// call our promise
var askMom = function () {
    willIGetNewPhone
    .then(showOff) // chain it here
    .then(function (fulfilled) {
            console.log(fulfilled);
         // output: 'Hey friend, I have a new black Samsung phone.'
        })
        .catch(function (error) {
            // oops, mom don't buy it
            console.log(error.message);
         // output: 'mom is not happy'
        });
};
```

Promises are Asynchronous

- Flattens the callbacks using then()



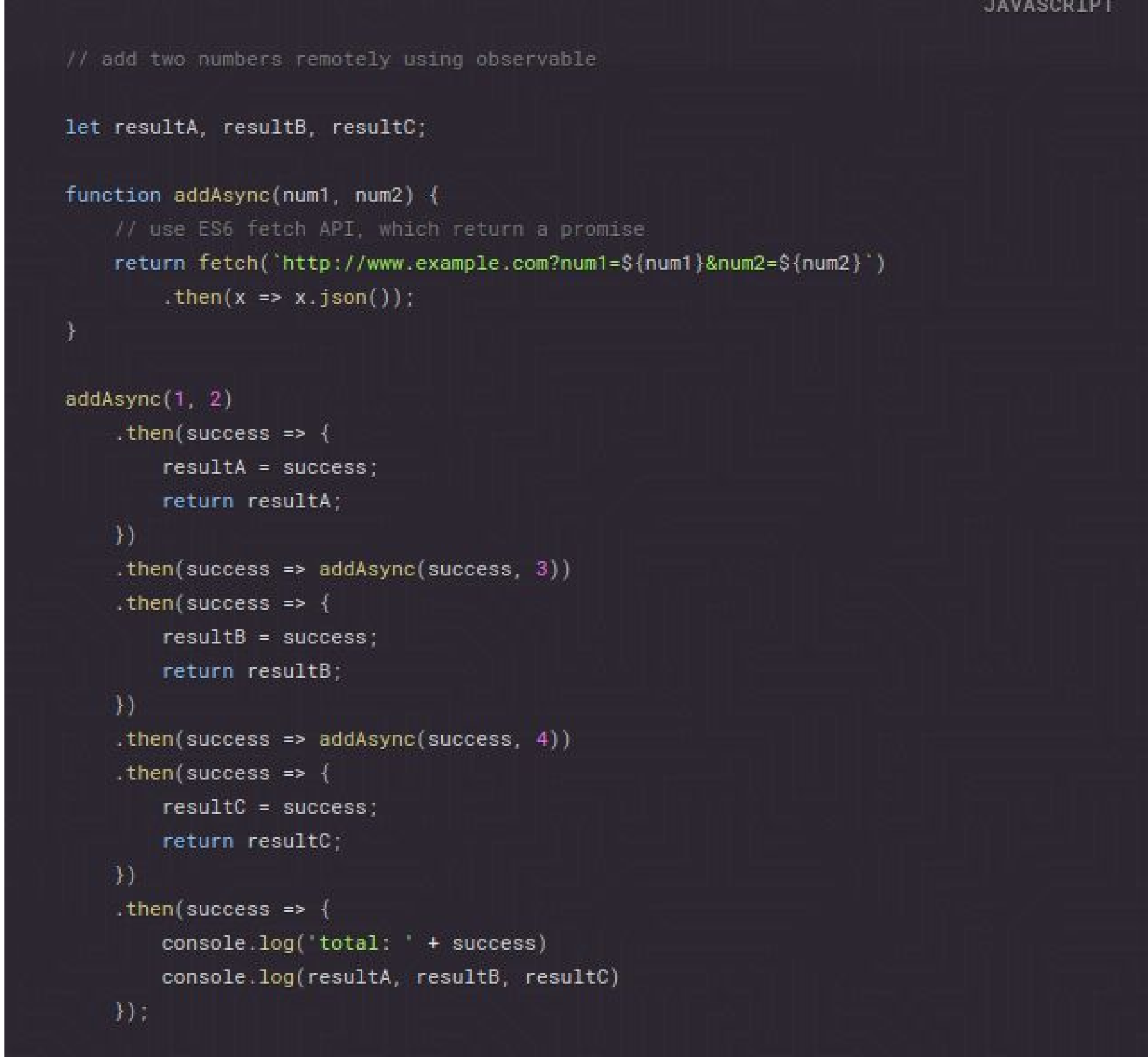
```javascript
// call our promise
var askMom = function () {
    console.log('before asking Mom'); // log before
    willIGetNewPhone
        .then(showOff)
        .then(function (fulfilled) {
            console.log(fulfilled);
        })
        .catch(function (error) {
            console.log(error.message);
        });
    console.log('after asking mom'); // log after
}
```



```
1. before asking Mom
2. after asking mom
3. Hey friend, I have a new black Samsung phone.
```

Using Promises

- Will be repeated at the end.
- Escape from callback hell.



```javascript
// add two numbers remotely using observable

let resultA, resultB, resultC;

function addAsync(num1, num2) {
    // use ES6 fetch API, which return a promise
    return fetch(`http://www.example.com?num1=${num1}&num2=${num2}`)
        .then(x => x.json());
}

addAsync(1, 2)
    .then(success => {
        resultA = success;
        return resultA;
    })
    .then(success => addAsync(success, 3))
    .then(success => {
        resultB = success;
        return resultB;
    })
    .then(success => addAsync(success, 4))
    .then(success => {
        resultC = success;
        return resultC;
    })
    .then(success => {
        console.log('total: ' + success)
        console.log(resultA, resultB, resultC)
    });
```