Promise

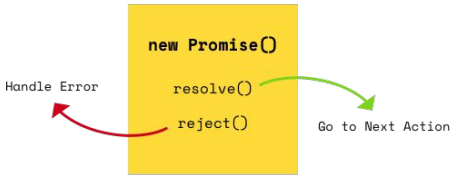
What is Promise?

"Imagine you are a kid. Your mom promises you that she'll get you a new phone next week."

You *don't know* if you will get that phone until next week. Your mom can either *really buy* you a brand new phone, or *stand you up* and withhold the phone if she is not happy :(.

That is a **promise**. A promise has 3 states. They are:

1. **Pending**: You *don't know* if you will get that phone
2. **Fulfilled**: Mom is happy, she buys you a brand new phone
3. **Rejected**: Your mom is not happy, she withholds the phone

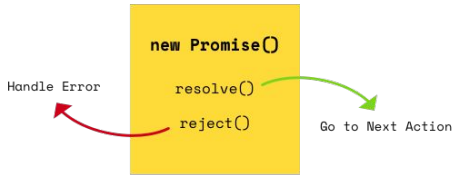


Credits to
https://scotch.io/tutorials/javascript-promises-for-dummies

Promise example - define your promise

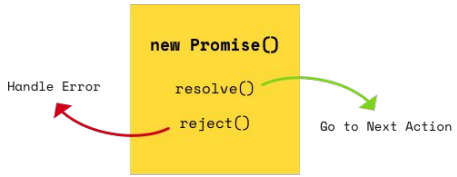
```
let isMomHappy = true;
// Promise
const willIGetNewPhone = new Promise(
    (resolve, reject) => { // fat arrow
        if (isMomHappy) {
            const phone = {
                brand: 'Samsung',
                color: 'black'
            };
            resolve(phone);
        } else {
            const reason = new Error('mom is not happy');
            reject(reason);
        }

    }
);
```



Promise example - call promise

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



Promise example - Chaining Promises

Let's say, you, the kid, **promises** your friend that you will **show them** the new phone when your mom buy you one.

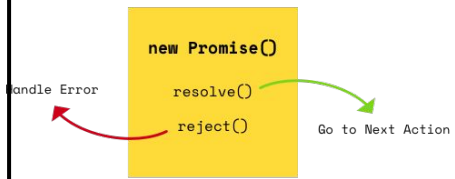
```
// 2nd promise
const showOff = phone => {
  return new Promise(
    (resolve, reject) => {
      let message = `Hey friend, I have a new ${phone.color} ${phone.brand} phone`;

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

Shorten syntax:

```
// Shorten:
const showOff = phone => {
      let message = `Hey friend, I have a new
${phone.color} ${phone.brand} phone`;

      return Promise.resolve(message);
};
```



Promise example - Full Code

```
const isMomHappy = true;

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

    }
);
```

```
const showOff = phone => {
      let message = `Hey friend, I have a new ${phone.color}
${phone.brand} phone`;

      return Promise.resolve(message);
};

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

askMom();
```

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

Promise example - Full Code - ES7 async await

Prettier and easier to understand - use **await** instead of .then and wrap with **try** and **catch** instead of .catch

```
// call our promise
async function askMom() {
  try {
    console.log('before asking Mom');

    let phone = await willIGetNewPhone;
    let message = await showOff(phone);

    console.log(message);
    console.log('after getting a response');
  } catch (error) {
    console.log(error.message);
  }
}
```

```
(async () => {
  await askMom();
})();
```

Every function that uses **await** needs to be defined as **async**

setTimeout example - by callback

Won't work!

```
const getUser = () => {
  setTimeout(() => {
    return { name: 'Max' }
  }, 2000)
}

// This doesn't actually fetch the user
const user = getUser()
console.log(user.name) // This won't work
```

Works great!

```
const getUser = cb => {
  setTimeout(() => {
    cb({ name: 'Max' })
  }, 2000)
}

getUser(user => {
  console.log(user.name) // Prints 'Max' after 2 seconds
})
```

The reason why it doesn’t work is that **JavaScript code runs in a non-blocking way.**
It won’t wait for async code to execute and return a value - it will only execute line for line until it’s done.

setTimeout example - callback hell

```
const checkAuth = cb => {
  // In reality, you of course don't have a timer but will probably reach out to a server
  setTimeout(() => {
    cb({ isAuth: true })
  }, 2000)
}
const getUser = (authInfo, cb) => {
  if (!authInfo.isAuth) {
    cb(null)
    return
  }
  setTimeout(() => {
    cb({ name: 'Max' })
  }, 2000)
}
```

```
checkAuth(authInfo => {
  getUser(authInfo, user => {
    console.log(user.name)
  })
})
```

This part called **"callback hell"** because it can grow very quickly to a big code of nested functions...

setTimeout example - promise!

```
const checkAuth = () => {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve({ isAuth: false });
    }, 2000);
  });
};
const getUser = (authInfo) => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (!authInfo.isAuth) {
        reject('ERROR: UNAUTHORIZED');
      }
      resolve({ name: 'Max' });
    }, 2000);
  });
};
```

```
checkAuth()
  .then(authInfo => {
    // returns a new promise which may use the authInfo we fetched
    return getUser(authInfo);
  })
  .then(user => {
    console.log(user.name); // prints the user name
  })
  .catch(error => {
    console.log(error); // prints error
  });
```

setTimeout example - promise by async await!

```
const checkAuth = () => {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve({ isAuth: false });
    }, 2000);
  });
};
const getUser = (authInfo) => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (!authInfo.isAuth) {
        reject('ERROR: UNAUTHORIZED');
      }
      resolve({ name: 'Max' });
    }, 2000);
  });
};
```

```
(async () => {
  try {
    let authInfo = await checkAuth();
    let user = await getUser(authInfo);
    console.log(user.name);
  } catch(error) {
    console.log(error);
  }
})();
```

loadScript example

```
function loadScript(src) {
  return new Promise((resolve, reject) => {
    let script = document.createElement('script');
    script.src = src;

    script.onload = () => resolve(script);
    script.onerror = () =>
     reject(new Error('Script load error: ' + src));

    document.head.append(script);
  });
}
```

```
let promise = loadScript(
  'https://cdnjs.cloudflare.com/ajax/libs/lodash.js/3.2.0/lodash.js'
);

promise.then(
  script => alert(`${script.src} is loaded!`),
  error => alert(`Error: ${error.message}`)
);
```

Remember the long XHR? Now we have fetch() !

```
fetch('./api/some.json')
  .then(
    function(response) {
      if (response.status !== 200) {
        console.log('Looks like there was a problem. Status Code: ' +
          response.status);
        return;
      }

      // Examine the text in the response
      response.json().then(function(data) {
        console.log(data);
      });
    }
  )
  .catch(function(err) {
    console.log('Fetch Error :-S', err);
  });
```

Promise Exercises

1. The built-in function setTimeout uses callbacks. Create a promise-based alternative.

   Use this code:

   ```
   function delay(ms) {
     // your code
   }

   delay(3000).then(() => alert('runs after 3 seconds'));
   ```

2. Write your own two chained promises

More Info

1. https://javascript.info
2. https://eloquentjavascript.net
3. https://www.academind.com/learn/javascript/callbacks-vs-promises-vs-rxjs-vs-async-awaits/
4. https://scotch.io/tutorials/javascript-promises-for-dummies