Promises

Promises - object that will produce a single value sometime in the future. Either a resolved value or reason it's not resolved.

Promise state:

- pending (unresolved)
- fulfilled (resolved)
- rejected

Can act kind of like event listeners that only run once.

Useful for asynchronous success/failure operations, such as API calls. We are less interested in the exact time it became available and more interested in reacting to the outcome.

So we are reacting to something that happens asynchronously.

For full explanation, see *PDF of Promise Theory*

Using Callbacks - nested functions

// Once element is clicked, run submitForm

```
el.addEventListener(“click”, submitForm);
```

// callback pyramid of doom

// move left, then right, then up

```
movePlayer(100, ‘Left’, function ( ) {
    movePlayer(10, ‘Right’, function ( ) {
        movePlayer(330, ‘Up’, function ( ) {
        });
    });
});
```

Same thing with Promises - chaining

```
movePlayer(100, ‘Left’)
    .then(( ) => movePlayer(10, ‘Right’))
    .then(( ) => movePlayer(330, ‘Up’))
```

Promise Example

// define the promise…

```
const promise = new Promise((resolve, reject) => {
    if (true) {
        resolve ('Stuff worked');
    } else {
        reject ('Error, it broke');
        }
})
```

```
> const promise = new Promise((resolve, reject) => {
      if (true) {
          resolve ('Stuff worked');
      } else {
          reject ('Error, it broke');
      }
  })
<· undefined
```

```
promise.then(result => console.log(result));
```

```
> promise.then(result => console.log(result));
  Stuff worked                                    VM307:1
<· ▶ Promise {<resolved>: undefined}
```

```
promise
```

```
> promise
<· ▶ Promise {<resolved>: "Stuff worked"}
```

```
promise.then(result => result + '!')
```

```
> promise
      .then(result => result + '!')
<· ▶ Promise {<resolved>: "Stuff worked!"}
```

```
promise
    .then(result => result + '!')
    .then(result2 => { console.log(result2); })
```

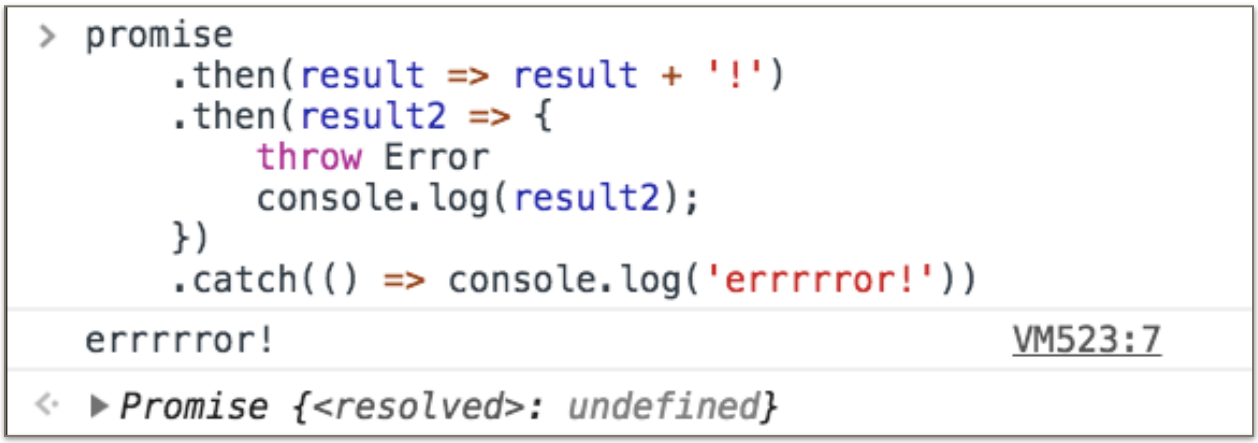
```
> promise
      .then(result => result + '!')
      .then(result2 => { console.log(result2); })
  Stuff worked!                                    VM497:3
```

Catching errors with .catch

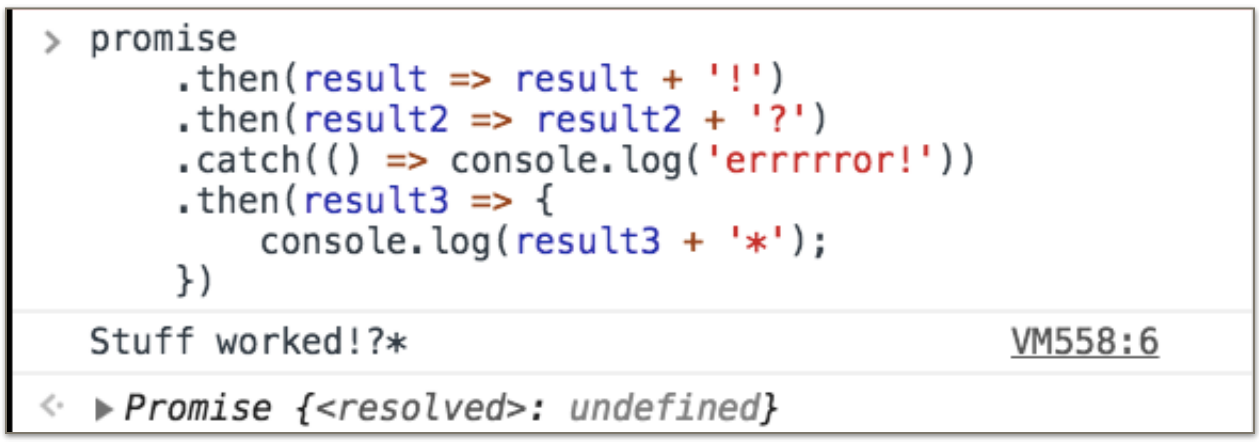
```
promise
    .then(result => result + '!')
    .then(result2 => {
        throw Error
        console.log(result2);
    })
```

```
> promise
      .then(result => result + '!')
      .then(result2 => {
          throw Error
          console.log(result2);
      })
  ◂ ▸Promise {<rejected>: f}
  ⊗ ▸Uncaught (in promise) f Error() {          (index):1
    [native code] }
```

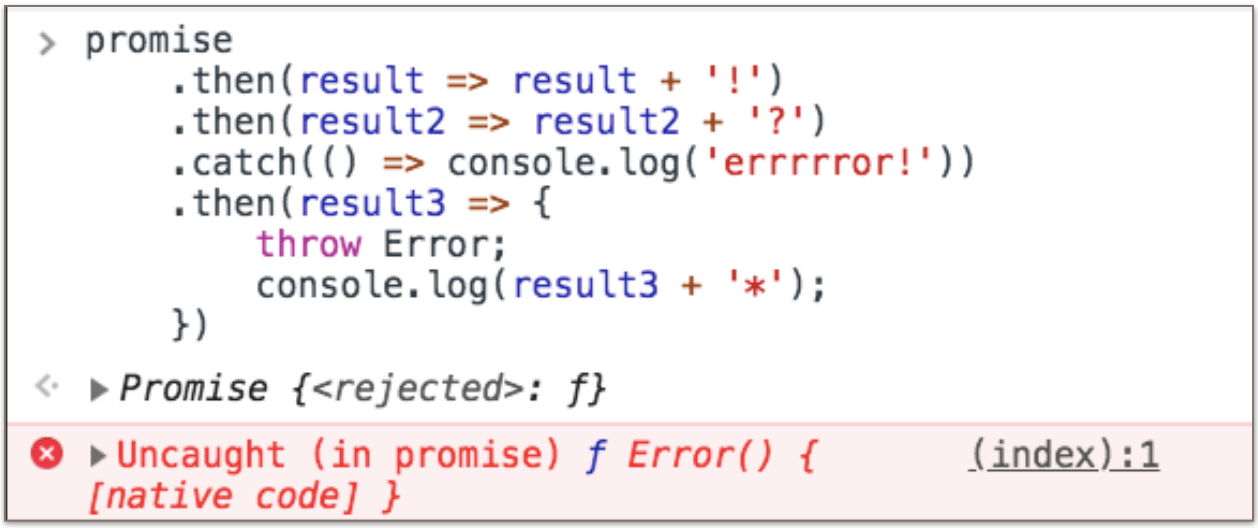
```
promise
    .then(result => result + '!')
    .then(result2 => {
        throw Error
        console.log(result2);
    })
    .catch(() => console.log(‘errrrror!’))
```




```
promise
    .then(result => result + '!')
    .then(result2 => result2 + ‘?’)
    .catch(() => console.log(‘errrrror!’))
    .then(result3 => {
        console.log(result3 + ‘*’);
    })
```



```
promise
    .then(result => result + '!')
    .then(result2 => result2 + '?')
    .catch(() => console.log('errrrror!'))
    .then(result3 => {
        throw Error;
        console.log(result3 + '*');
    })
```



// CATCH only works on what is before it

setTimeout and Promise.all

```
const promise2 = new Promise((resolve, reject) => {
    setTimeout(resolve, 100, “HIIIIIII”)
})

const promise3 = new Promise((resolve, reject) => {
    setTimeout(resolve, 1000, “BYEEEE”)
})

const promise4 = new Promise((resolve, reject) => {
    setTimeout(resolve, 3000, “FINAL ANSWR”) //
wait 3s to resolve
})
```

```
> const promise2 = new Promise((resolve, reject) => {
      setTimeout(resolve, 100, 'HIIIIIII')
  })
< undefined
> const promise3 = new Promise((resolve, reject) => {
      setTimeout(resolve, 1000, 'BYEEEE')
  })

  const promise4 = new Promise((resolve, reject) => {
      setTimeout(resolve, 3000, 'FINAL ANSWER')
  })
< undefined
```

```
Promise.all([promise2, promise3, promise4])
    .then(values => {
        console.log(values);
    })
```

```
  Promise.all([promise2, promise3, promise4])
      .then(values => {
          console.log(values);
      })
< ▶ Promise {<pending>}
                                                VM731:16
▶ (3) ["HIIIIIII", "BYEEEE", "FINAL ANSWER"]
```

Chaining Promises with .then

```
// stack overflow example // // // // //

function foo(){ // foo is promise function that returns 'Value'
when resolved
    return Promise.resolve("Value");
}

foo().then(console.log); // 'Value' passed to console.log()

function task2(e){
    console.log("In two got " + e);
    return " Two ";
}
function task3(e){
    console.log("In three got " + e);
    return " Three ";
}

foo().then(task2);
foo().then(task3);
foo().then(task2).then(task3) // chaining
```

```
// Steph Example // // // // //

// create function that returns a promise
function stephPromise() {
    return Promise.resolve('Apple');
}

function hungry(e){
    console.log("Eat the " + e);
    return "Full";
}

function mood(e){
    console.log("Mood is " + e);
    return "sleep";
}

stephPromise().then(hungry).then(mood);
stephPromise().then(hungry).then(mood).then(console.log);
stephPromise().then(hungry).then(hungry).then(console.log)

stephPromise().then(hungry);
stephPromise().then(mood);
```

```
// ALL EXAMPLE (an extra example with diff syntax

// .all() example  // // // // //   // // // // //
var requestComplete = true;

promise1 = new Promise((resolve, reject) => {
  if (requestComplete)
    resolve("data received from 1");
})

promise2 = new Promise((resolve, reject) => {
  if (requestComplete)
    resolve("data received from 2");
})

promise3 = new Promise((resolve, reject) => {
  setTimeout(()=>{
  resolve("data received from 3");
  }, 2000);
})

Promise.all([promise1, promise2, promise3])
    .then( (message) => { console.log(message); })
```

```
// andrei example with URLs

const urls = [
    'https://jsonplaceholder.typicode.com/users',
    'https://jsonplaceholder.typicode.com/posts',
    'https://jsonplaceholder.typicode.com/albums'
]

Promise.all(urls.map(url => {
    return fetch(url)
    .then(resp => resp.json())
}))
    .then(results => {
    console.log(results[0])
    console.log(results[1])
    console.log(results[2])
    })
    .catch(() => console.log('error!'))
```

```
// map since array

// for each URL, we want fetch (API) to make ajax call to URL

// & then run result thru response.json (convert JSON to readable format)

// then console log the results

// catch errors
```

```
▶ Promise {<pending>}

                                                    VM912:12
▶ (10) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}
  , {…}]
                                                    VM912:13
  (100) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
▶ {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}]
                                                    VM912:14
  (100) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
▶ {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}]
```