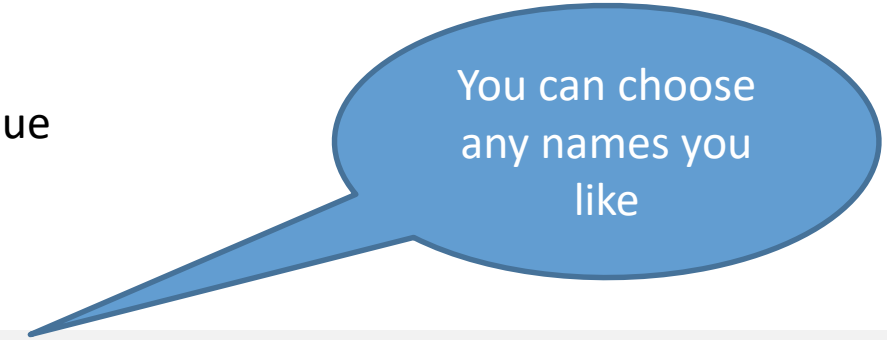
Promises

A Promise represents an operation that hasn't completed yet, but is expected in the future

Syntax

```
new Promise( /* executor */ function(resolve, reject) {
 ...
});
```

- The executor function is executed immediately by the Promise implementation which provides the resolve and reject functions
- The resolve and reject functions are bound to the promise and calling them fulfills or rejects the promise
- The executor is expected to initiate some asynchronous work and then, once that completes:
  - Either call the resolve function
    - To resolve the promise's final value
  - Or call the reject function
    - To reject it if an error occurred




```
new Promise(function(succes, fail) { ... });
```

Promise.prototype.**then**()

- The then() method returns a Promise and takes two arguments:
  1. Callback function for the success
  2. Callback function for failure cases

- Syntax:

```
p.then(onFulfilled, onRejected);
```

  - onFulfilled
    - A function called when the Promise is fulfilled (succes)
    - This function has one argument, the fulfillment value
  - onRejected
    - A function called when the Promise is rejected (error)
    - This function has one argument, the rejection reason

Using the then method

```
var p1 = new Promise(function (resolve, reject) {
    // Do some (asynchronous) work – e.g. make an ajax call
    resolve("Success!");
    // or
    // reject ("Error!");
});

p1.then(function (value) {
    console.log(value); // Success!
}, function (reason) {
    console.log(reason); // Error!
});
```

Promise.prototype.catch()

- The catch() method returns a Promise and deals with rejected cases only
- It behaves the same as calling:

```
Promise.prototype.then(undefined, onRejected)
```

- Syntax:

```
p.catch(onRejected);
```

  - onRejected
    - A Function called when the Promise is rejected
    - This function has one argument, the rejection reason

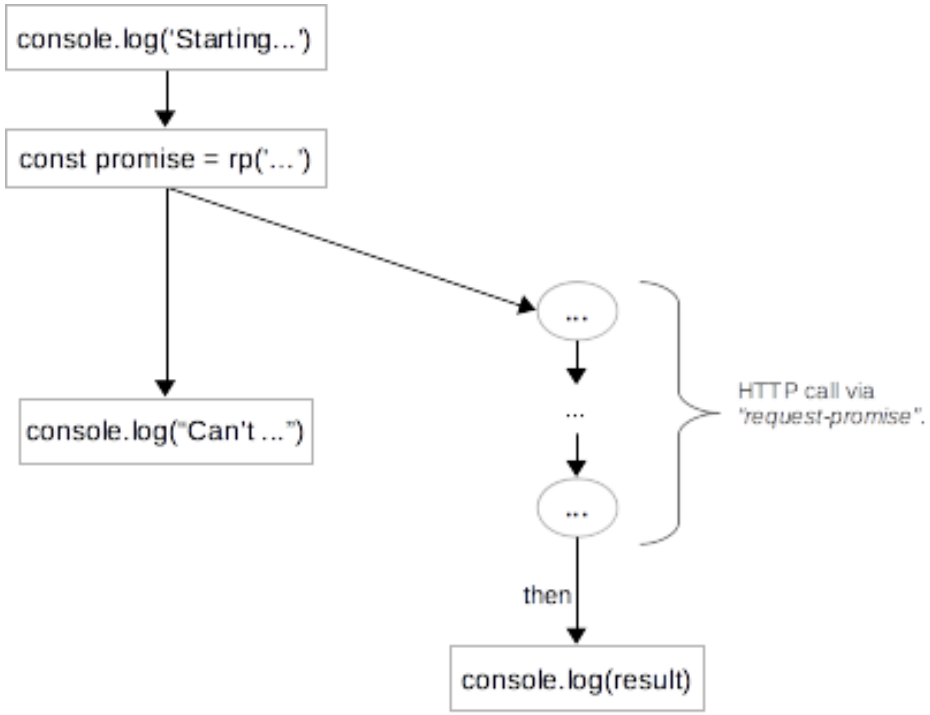
Using catch

```
var p1 = new Promise(function (resolve, reject) {
    resolve('Success');
});

p1.then(function (value) {
    console.log(value); // "Success!"
    return Promise.reject('oh, no!');
}).catch(function (e) {
    console.log(e); // "oh, no!"
}).then(function () {
    console.log('after a catch the chain is restored');
}, function () {
    console.log('Not fired due to the catch');
});
```

Computational process of a promise

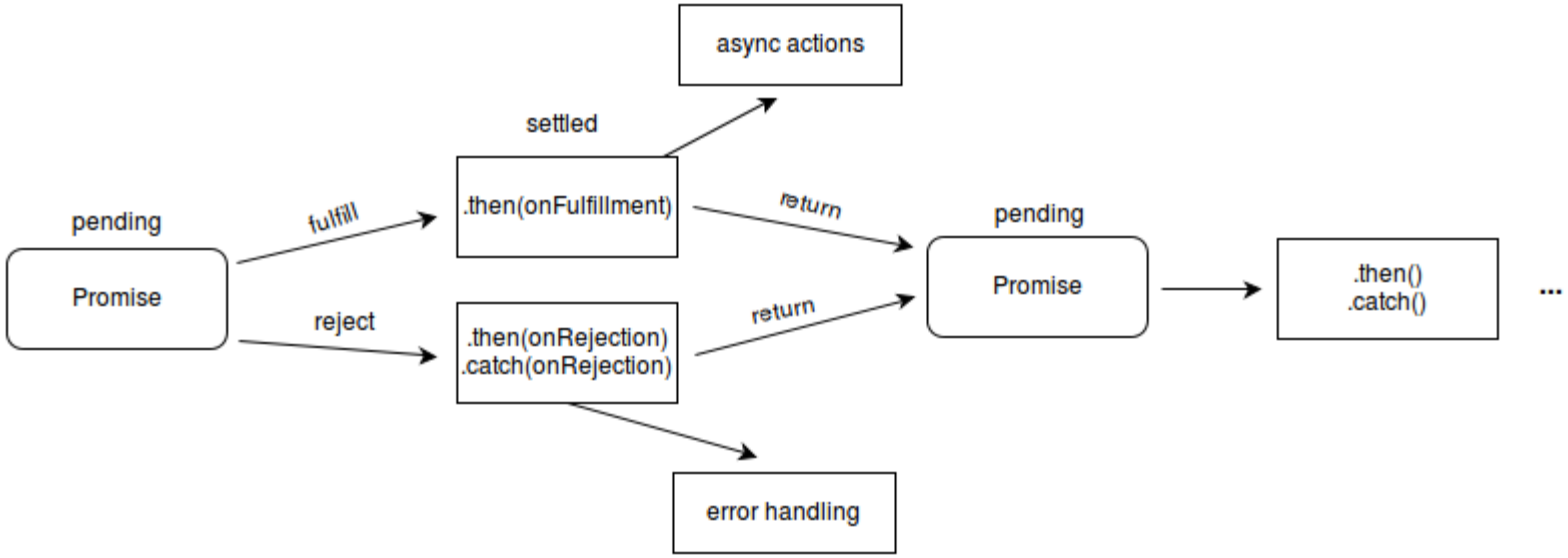
```
console.log('Starting Execution');
const promise = rp('http://example.com/'); // Returns a Promise
promise.then(result => console.log(result));
console.log("Can't know if promise has finished yet...");
```



The only way to schedule code after a promise is to specify a callback via the "then" method.

State Chart

- As the Promise.prototype.then() and Promise.prototype.catch() methods return promises, they can be chained



Chaining

```
var p2 = new Promise(function (resolve, reject) {
    resolve(1);
});

p2.then(function (value) {
    console.log(value); // 1
    return value + 1;
}).then(function (value) {
    console.log(value); // 2
});

p2.then(function (value) {
    console.log(value); // 1
});
```

Promise.resolve(x)

- The resolve function returns either a new promise resolved with the passed argument, or the argument itself if the argument is a promise produced by this constructor

Promise.reject(reason)

- Returns a Promise object that is rejected with the given reason

Promise.all(iterable)

- The all function returns a new promise which is fulfilled with an array of fulfillment values for the passed promises, or rejects with the reason of the first passed promise that rejects
- It resolves all elements of the passed iterable to promises as it runs this algorithm

Example – the Callee

```
function get(url) {
    return new Promise(function (succeed, fail) {
        var req = new XMLHttpRequest();
        req.open("GET", url, true);
        req.addEventListener("load", function () {
            if (req.status < 400)
                succeed(req.responseText);
            else
                fail(new Error("Request failed: " +
                               req.statusText));
        });
        req.addEventListener("error", function () {
            fail(new Error("Network error"));
        });
        req.send(null);
    });
}
```

The executor function

Example – the Caller

Returns a Promise

```
get("files/data.txt").then(function (text) {
    // Resolve / succeed
    console.log("data.txt: " + text);
}, function (error) {
    // Reject / fail
    console.log ("Failed to fetch data.txt: " + error);
});
```

References & Links

- ES6 Promises in Depth
  https://ponyfoo.com/articles/es6-promises-in-depth

- MDN Promise
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

- **Await and Async Explained with Diagrams and Examples**
  http://nikgrozev.com/2017/10/01/async-await/