PRACTICAL PROLONGED PROCESS PROGRAMMING

a prompt promise primer

- I want to read the contents of fileA.txt
- Then, once that's done, I want to read the contents for fileB.txt
- If either of them throw an error, I want to catch it

WILL THIS WORK?

```
fs.readFile("fileA.txt", function (err, data) {
  console.log(data);
});

fs.readFile("fileB.txt", function (err, data) {
  console.log(data);
});
```

NESTED CALLBACKS

```
fs.readFile("fileA.txt", function (err, data) {
  console.log(data);

  fs.readFile("fileB.txt", function (err, data) {
    console.log(data);
  });
});
```

ERROR HANDLING

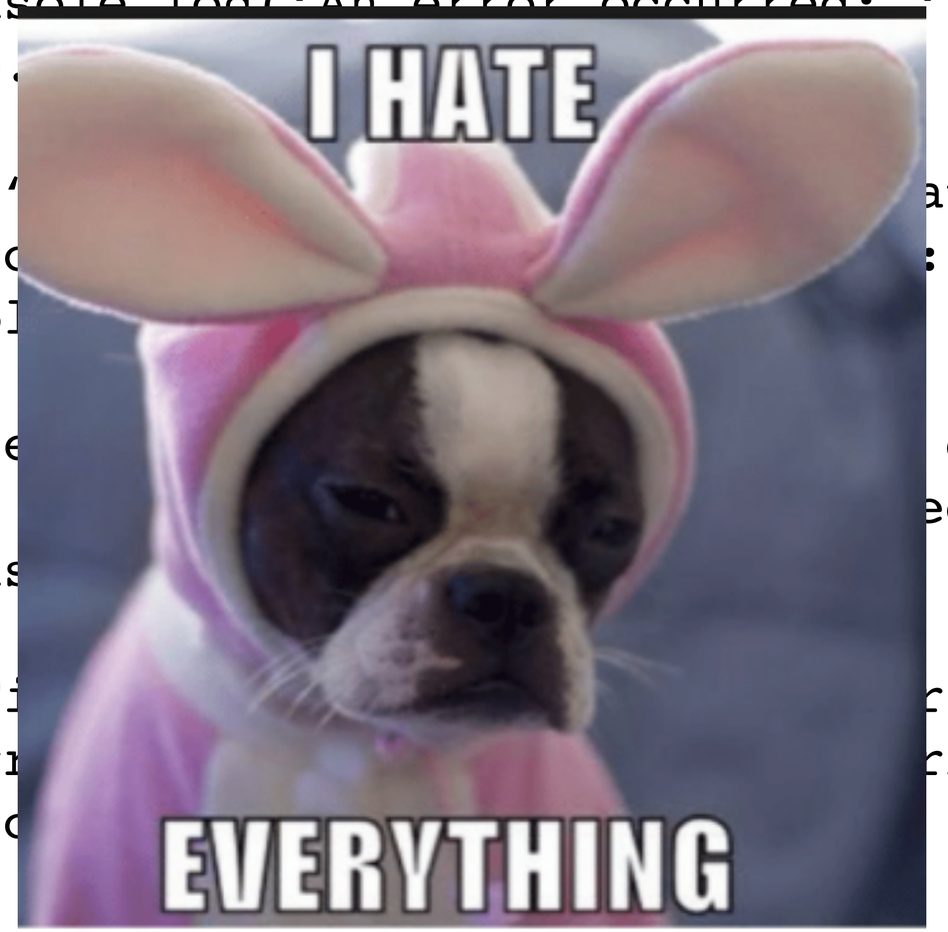
```
fs.readFile("fileA.txt", function (err, data) {
  if (err) console.log('An error occurred: ', err);
  else console.log(data);

  fs.readFile("fileB.txt", function (err, data) {
    if (err) console.log('An error occurred: ', err);
    else console.log(data);
  });
});
```

CALLBACK HELL

```
fs.readFile("fileA.txt", function (err, data) {
  if (err) console.log('An error occurred: ', err);
  else console.log(data);

  fs.readFile("fileB.txt", function (err, data) {
    if (err) console.log('An error occurred: ', err);
    else console.log(data);

    fs.readFile("fileC.txt", function (err, data) {
      if (err) console.log('An error occurred: ', err);
      else console.log(data);
    });
  });
});
```

```
fs.readFile("fileA.txt", function (err, data) {
  if (err) console.log('An error occurred: ', err);
  else console.log(data);

  fs.readFile("fileB.txt", function (err, data) {
    if (err) console.log('An error occurred: ', err);
    else console.log(data);

    fs.readFile("fileC.txt", function (err, data) {
      if (err) console.log('An error occurred: ', err);
      else console.log(data);

      fs.readFile("fileD.txt", function (err, data) {
        if (err) console.log('An error occurred: ', err);
        else console.log(data);
      });
    });
  });
});
```

```
fs.readFile("fileA.txt", function (err, data) {
  if (err) consol                                  ', err);
  else console.

  fs.readFile('                                    ata) {
    if (err) co                                    : ', err);
    else consol

    fs.readFile                                    data) {
      if (err)                                     ed: ', err);
      else cons

      fs.readFi                                    r, data) {
        if (err                                    rred: ', err);
        else co
      });
    });
  });
});
```



👆 current mood

WHAT IS A CALLBACK?

WHAT IS A CALLBACK?

Technically: a function passed to another function

two flavors…

- **Blocking**
- **Non-blocking**

BLOCKING CALLBACKS

*think: **portable code***

predicates

e.g. `arr.filter(function predicate (elem) {…});`

comparators

e.g. `arr.sort(function comparator (elemA, elemB) {…});`

iterators

e.g. `arr.map(function iterator (elem) {…});`

NON-BLOCKING CALLBACKS

*think: **control flow***

WHAT IS A CALLBACK?

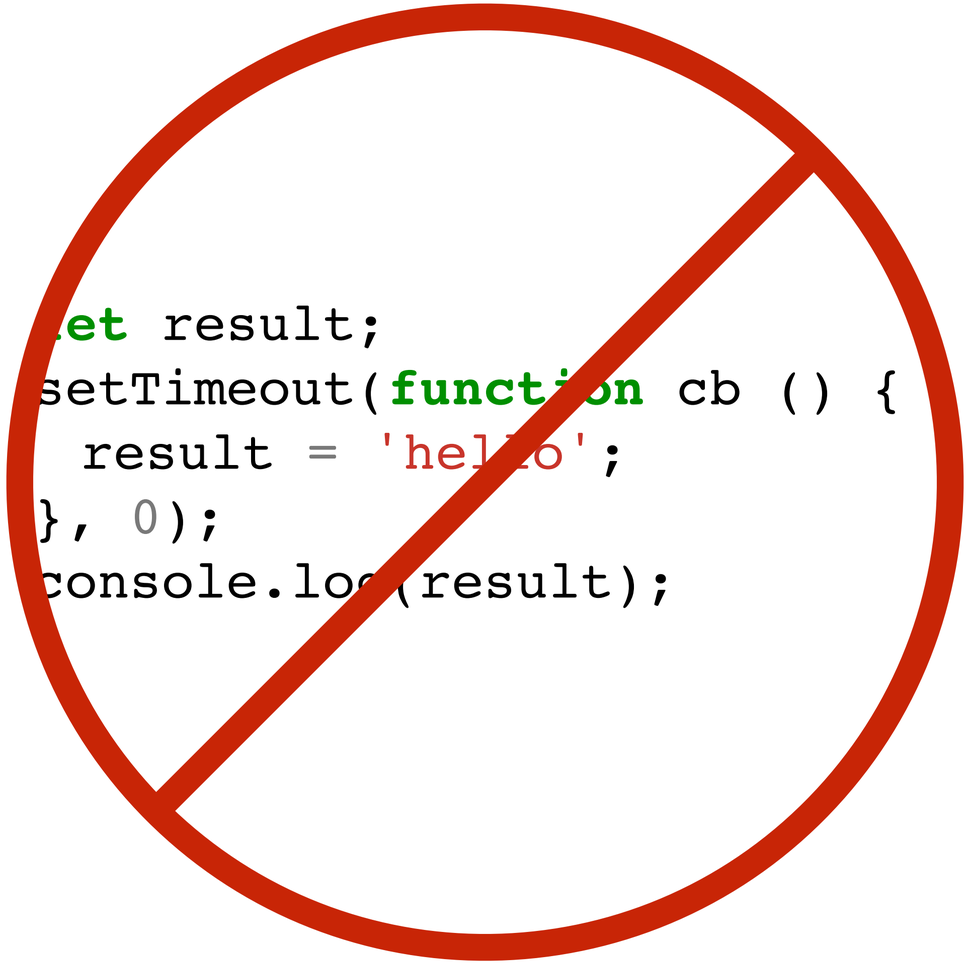
Technically: a function passed to another function

two flavors…

- Blocking
- Non-blocking

- event handler
- middleware
- vanilla async

LIKE THIS?



```
let result;
setTimeout(function cb () {
  result = 'hello';
}, 0);
console.log(result);
```

LIKE THIS?

```
let result = setTimeout(function cb () {
  return 'hello';
}, 0);
console.log(result);
```

LIKE THIS?

```
setTimeout(function cb () {
  let result = 'hello';
  console.log(result);
}, 0);
```

PROMISE

"A promise represents the eventual result of an asynchronous operation."

— THE PROMISES/A+ SPEC

PROMISES

- A Promise is a JavaScript object that follows a specific set of rules (the Promises/A+ spec)
- A Promise contains ("is for") some data that you get asynchronously (ex. setTimeout, fs.readFile, etc…)

HOW TO GET THAT ASYNC DATA

- A Promise implements a method called *then (Promise.prototype.then)*
- *Promise.prototype.then* accepts two callback functions as arguments (one for success, and one for errors)
- Once the Promise has the data (or an error), it will invoke those callback functions with the data (or an error)

CALLBACK V PROMISE

vanilla async callback

```
fs.readFile('file.txt',
  function callback (err, data) {…}
);
```

async promise

```
const promiseForData = fs.readFileAsync('file.txt')

promiseForData
.then(
  function onSuccess (data) {…},
  function onError (err) {…}
);
```

LIKE THIS?

```
let result;
promisifiedSetTimeout(0)
.then(function success () {
  result = 'hello';
});
console.log(result);
```

LIKE THIS?

```
let result = promisifiedSetTimeout(0)
.then(function success () {
  return 'hello';
});
console.log(result);
```

LIKE THIS?

```
promisifiedSetTimeout(0)
.then(function success () {
  let result = 'hello';
  console.log(result);
});
```

OKAY, LIKE, SO WHAT? WE STILL NEED CALLBACKS...

PORTABLE

```
let result;

fs.readFile(`file.txt`, function (err, file) {
  result = file;
});

doSomething(result) // what's result here?

module.exports = result; // LOL, just no…
```

PORTABLE

```
const promise = fs.readFileAsync('file.txt');

// call some other function on it
doSomething(promise);

// export the promise, use it elsewhere!
module.exports = promise;
```

MULTIPLE HANDLERS

```
const promise = fs.readFileAsync('file.txt');

// do one thing when it finishes
promise
.then(function (fileContents) {
  console.log(fileContents);
});

// do another thing when it finishes
promise
.then(function () {
  fs.unlink('file.txt');
});
```

LINEAR/FLAT

```
fs.readFile('fileONE.txt', function (err, data) {
  console.log('I just read file one');

  fs.readFile(`fileTWO.txt`, function (err, data) {
    console.log('I just read file two');
  });
});
```

LINEAR/FLAT

```
fs.readFileAsync('fileONE.txt')
.then(function () {
  console.log('I just read file one');

  fs.readFileAsync('fileTWO.txt')
  .then(function () {
    console.log('I just read file two');
  });
})
```

LINEAR/FLAT

```
fs.readFileAsync('fileONE.txt')
.then(function () {
  console.log('I just read file one');
  return fs.readFileAsync('fileTWO.txt');
})
.then(function () {
  console.log('I just read file two');
});
```

UNIFIED ERROR HANDLING

```
fs.readFile('fileONE.txt', function (err, data) {
  if (err) console.log(`Error with fileONE`);
  else console.log(`I just read fileONE`);

  fs.readFile(`fileTWO.txt`, function (err, data) {
    if (err) console.log(`Error with fileTWO`);
    else console.log(`I just read fileTWO`);

    fs.readFile(`FileTHREE.txt`, function (err, data) {
      // KILL ME NOW :(
    });
  });
});
```

UNIFIED ERROR HANDLING

```
fs.readFileAsync('fileONE.txt')
.then(function () {
  console.log('I just read file one');
  return fs.readFileAsync('fileTWO.txt');
})
.then(function () {
  console.log('I just read file two');
  return fs.readFileAsync(`fileTHREE.txt');
})
.then(null, function (err) { // or `.catch()`
  console.log('An error occurred at some point');
  console.log(err);
});
```

UNIFIED ERROR HANDLING

```
fs.readFileAsync('fileONE.txt')
.then(function () {
  console.log('I just read file one');
  return fs.readFileAsync('fileTWO.txt');
})
.then(function () {
  console.log('I just read file two');
  return fs.readFileAsync(`fileTHREE.txt');
})
.catch(function (err) {
  console.log('An error occurred at some point');
  console.log(err);
});
```

READING A FILE

SYNCHRONOUS

```
var path = 'demo-poem.txt';
console.log('- I am first -');
try {
  var buff = fs.readFileSync(path);
  console.log(buff.toString());
} catch (err) {
  console.error(err);
}
console.log('- I am last -');
```

ASYNC (CALLBACKS)

```
var path = 'demo-poem.txt';
fs.readFile(path, function (err, buff) {
  if (err) console.error(err);
  else console.log(buff.toString());
  console.log('- I am last -');
});
console.log('- I am first -');
```

ASYNC (PROMISES)

```
var path = 'demo-poem.txt';
promisifiedReadFile(path)
.then(function (buff) {
  console.log(buff.toString());
}, function (err) {
  console.error(err);
})
.then(function () {
  console.log('- I am last -');
});
console.log('- I am first -');
```

```javascript
const path = ‘demo-poem.txt';

promisifiedReadFile(path)
.then(function (buff) {
  console.log(buff.toString());
}, function (err) {
  console.error(err);
})
.then(function () {
  console.log('- I am last -');
});

console.log('- I am first -');
```

```javascript
const path = ‘demo-poem.txt';

fs.readFile(path, function (err, buff) {
  if (err) console.error(err);
  else console.log(buff.toString());
  console.log('- I am last -');
});

console.log('- I am first -');
```

```javascript
const path = ‘demo-poem.txt';

promisifiedReadFile(path)
.then(buff => {
  console.log(buff.toString());
}, err => {
  console.error(err);
})
.then(() => {
  console.log('- I am last -');
});

console.log('- I am first -');
```

```javascript
const path = ‘demo-poem.txt';

fs.readFile(path, (err, buff) => {
  if (err) console.error(err);
  else console.log(buff.toString());
  console.log('- I am last -');
});

console.log('- I am first -');
```

```javascript
const path = ‘demo-poem.txt';

promisifiedReadFile(path)
.then(buff => console.log(buff.toString()),
      err => console.error(err))
.then(() => console.log('- I am last -‘));

console.log('- I am first -');
```

```javascript
const path = ‘demo-poem.txt';

fs.readFile(path, (err, buff) => {
  if (err) console.error(err);
  else console.log(buff.toString());
  console.log('- I am last -');
});

console.log('- I am first -');
```

```javascript
const path = ‘demo-poem.txt';

promisifiedReadFile(path)
.then(buff => console.log(buff.toString()))
.then(() => console.log('- I am last -‘))
.catch(err => console.error(err));

console.log('- I am first -');
```

```javascript
const path = ‘demo-poem.txt';

fs.readFile(path, (err, buff) => {
  if (err) console.error(err);
  else console.log(buff.toString());
  console.log('- I am last -');
});

console.log('- I am first -');
```

PROMISE ADVANTAGES

- Portable
- Multiple handlers
- “Linear” or “flat” chains
- Unified error handling

IMPLEMENTATIONS

- Adehun
- avow
- ayepromise
- bloodhound
- bluebird
- broody-promises
- CodeCatalyst
- Covenant
- D
- Deferred
- fate
- ff
- FidPromise
- ipromise
- Legendary
- Lie
- microPromise
- mpromise
- Naive Promesse
- Octane
- ondras
- potch
- P
- Pacta
- Pinky
- PinkySwear
- Potential
- promeso
- promiscuous
- Promis
- Promix
- Promiz
- Q
- rsvp
- Shvua
- Ten.Promise
- then
- ThenFail
- typescript-deferred
- vow
- when
- yapa
- yapi
- Zousan