HELLO!

ASYNCHRONOUS JAVASCRIPT

Allow work to happen in the background

Get a callback when it's done

ASYNCHRONOUS JAVASCRIPT

```
$.get("/foo/bar", function(data) {
  // ...
}
```

ASYNCHRONOUS JAVASCRIPT

```
fs.stat("/etc/foobar", function(err, stats) {
  // ...
}
```

ASYNCHRONOUS JAVASCRIPT

Good for performance

Hard to reason about

WHAT TO WORRY ABOUT

NOT "Pyramid of Doom"

PYRAMID OF DOOM

```
db.table('unprocessed_files').first(function(filename) {
  fs.stat(filename, function(err, stat) {
    if (stats.isFile()) {
      fs.readFile(filename, function(err, data) {
        var processed = process(data);
        fs.writeFile('/var/processed/' + filename, processed, funct
          if (!err) {
            db.table('unprocessed_files').delete(filename, function
              response.writeHead(200);
              response.finish();
            });
          }
        });
      });
    }
  });
```

PLAINS OF DOOM

```
var filename = db.table('unprocessed_files').first();
var stats = fs.stat(filename);
if (stats.isFile()) {
  var data = fs.readFile(filename);
  var processed = process(data);
  var err = fs.writeFile('/var/processed/' + filename, processed);
  if (!err) {
    db.table('unprocessed_files').delete(filename);
    response.writeHead(200);
    response.finish();
  }
}
```

LOTS OF TINY PYRAMIDS

```
function processAndWriteToFile(data, callback) {
  var processed = process(data);
  fs.writeFile('/var/processed/'+filename, processed, callback);
}

function readAndProcessFile(filename, callback) {
  fs.stat(filename, function(err, stat) {
    if (stats.isFile()) {
      fs.readFile(filename, callback);
    }
  });
}
```

```
function dbPopAndHandle(table, processingFn, callback) {
  db.table(table).first(function(record) {
    processingFn(record, function(err) {
      db.table(table).delete(filename, callback);
    });
  });
}

function popAndProcessFile(callback) {
  dbPopAndHandle('unprocessed_files', function(record) {
    readAndProcessFile(record, function(err, data) {
      processAndWriteToFile(data, callback);
    });
  });
}
```

```
popAndProcessFile(function(err) {
  if (!err) {
    response.writeHead(200);
    response.finish();
  }
});
```

DON'T WORRY ABOUT PYRAMID OF DOOM

WORRY ABOUT OTHER, MORE IMPORTANT THINGS

HERE'S WHAT YOU SHOULD FEAR

```
dbAccess ----\
              ---> collate(d1,d2) ------\
dbAccess ----/                           ----> output(c1, n1)
                                        /
network -------------------------------/
```

THE CRUCIBLES

CRUCIBLE #1

A single async operation.

```
dbAccess ---> output(d1)
```

CRUCIBLE #2

Multiple parallel async operations.

```
dbAccess -----\
dbAccess ----\ \
dbAccess ---------> output(d1,d2,d3,d4,d5)
dbAccess ----/ /
dbAccess -----/
```

CRUCIBLE #3

Parallel & serial operations, combined.

```
dbAccess ----\
              ---> collate(d1,d2) ------\
dbAccess ----/                           ----> output(c1, n1)
                                        /
network --------------------------------/
```

TEST API

input: function([string]crucibleNum)

TEST API

output: function([any]args...)

errored: function()

TEST API

dbAccess: function([int]id, callback(err, [obj]data))

TEST API

collate: function([obj]d1, [obj]d2, callback(err, [string]result))

TEST API

network: function([int]id, callback(err, [obj]data))

THE CONTENDERS

1. Vanilla JavaScript
2. Async.js
3. Promises
4. IcedCoffeeScript

NOT MENTIONED: ES6

- I don't have personal experience with it
- Usable in front ends for real users: approximately **never**

LET'S BEGIN

CRUCIBLE 1

VANILLA JAVASCRIPT

```javascript
var  c = require("../crucibles");
var id = c.input("1");

c.dbAccess(id, function(err, data) {
  if (err) {
    c.errored();
  } else {
    c.output(data);
  }
});
```

CRUCIBLE 1

ASYNC.JS

```
var  c = require("../crucibles");
var id = c.input("1");

c.dbAccess(1, function(err, data) {
  if (err) {
    c.errored();
  } else {
    c.output(data);
  }
});
```

CRUCIBLE 1

PROMISES

```
dbAccess: function(id) {
  var deferred = Q.defer();
  c.dbAccess(id, function(err, data) {
    if (err) {
      deferred.reject(new Error(err));
    } else {
      deferred.resolve(data);
    }
  });
  return deferred.promise;
}
```

```
var c_p = require("./crucibles_promise_api");
var id = c_p.input("1");

c_p.dbAccess(1).then(
    function(data) {
      c_p.output(data);
    },
    function(err) {
      c_p.errored();
    }
  );
```

CRUCIBLE 1

ICEDCOFFEESCRIPT

```
c = require("../crucibles")

await c.dbAccess(1, defer err, data)
if err
  c.errored()
else
  c.output data
```

CRUCIBLE 2

VANILLA JAVASCRIPT

```javascript
var results = [];
var waiting = 0;
for (i=0; i<ids.length; i+=1) {
  waiting += 1;
  c.dbAccess(ids[i], function(data) {
    results[i] = data;
    // ...
  });
}

// HAHAHA NOPE
```

```
var results = [];
var hadError = false;
var waiting = 0;
for (i=0; i<ids.length; i+=1) {
  (function(i) {
    // ...
  })(i);
}
```

```
    waiting += 1;
    c.dbAccess(ids[i], function(err, data) {
      if (err) {
        hadError = true;
      }
      results[i] = data;
      waiting -= 1;
      if (waiting === 0) {
        if (hadError) {
          c.errored();
        } else {
          c.output(results);
        }
      }
    });
```

CRUCIBLE 2

ASYNC.JS

```javascript
var async = require("async");

async.map(ids, c.dbAccess, function(err, results) {
  if (err) {
    c.errored();
  } else {
    c.output(results);
  }
});
```

CRUCIBLE 2

PROMISES

```
var Q = require("q");

var promises = ids.map(c_p.dbAccess);
Q.all(promises).then(
    function(data) {
      c_p.output(data);
    },
    function(err) {
      c_p.errored();
    }
  );
```

CRUCIBLE 2

ICEDCOFFEESCRIPT

```
results = []
errors = []
await
  for id, i in ids
    c.dbAccess(id, defer errors[i], results[i])

errors = (err for err in errors when err?)
if errors.length isnt 0
  c.errored()
else
  c.output results
```

CRUCIBLE 3

VANILLA JAVASCRIPT

PREPARE YOURSELVES

```
var dbResults = [];
var dbWaiting = 0;
var networkWaiting = false;
var networkResult;
var collateWaiting = false;
var collateResult;
for (i=0; i<ids.length; i+=1) {
  (function(i) {
    dbWaiting += 1;
    c.dbAccess(ids[i], function(data) {
      dbResults[i] = data;
      dbWaiting -= 1;
      if (dbWaiting === 0) {
        collateWaiting = true;
        c.collate(dbResults[0], dbResults[1], function(c1) {
          collateWaiting = false;
          collateResult = c1;
          if (!networkWaiting) {
            c.output(collateResult, networkResult);
          }
        });
      }
    });
  })(i);
}
networkWaiting = true;
```

```
});
```

```
var getDbResults = function(callback) {
  // ...
  for (i=0; i<ids.length; i+=1) {
      // ...
      c.dbAccess(ids[i], function(err, data) {
        // ...
        if (dbWaiting === 0) {
          if (errors.length > 0) {
            callback(errors);
          } else {
            callback(null, dbResults);
          }
        }
      });
  }
};
```

```javascript
var getAndCollate = function(callback) {
  getDbResults(function(err, dbResults) {
    if (err) {
      callback(err);
    } else {
      c.collate(dbResults[0], dbResults[1], callback);
    }
  });
};
```

```
var networkAndCollation = function(callback) {
  var networkWaiting;
  var networkResult;
  var collateWaiting;
  var collateResult;
  var errors = [];
  var tryContinue = function() {
    if (!networkWaiting && !collateWaiting) {
      if (errors.length > 0) {
        callback(errors);
      } else {
        callback(null, [collateResult, networkResult]);
      }
    }
  };
  // Continued...
  // ...
  collateWaiting = true;
  getAndCollate(function(err, c1) {
    if (err) {
      errors.push(err);
    }
    collateWaiting = false;
    collateResult = c1;
    tryContinue();
  });
```

```
  // ...
  networkWaiting = true;
  c.network(1, function(err, n1) {
    if (err) {
      errors.push(err);
    }
    networkWaiting = false;
    networkResult = n1;
    tryContinue();
  });
};
```

```
networkAndCollation(function(err, data) {
  if (err) {
    c.errored();
  } else {
    c.output(data[0], data[1]);
  }
});
```

CRUCIBLE 3

ASYNC.JS

```
var getDbResults = function(callback) {
  async.map(ids, c.dbAccess, callback);
};
```

```
var getAndCollate = function(callback) {
  async.waterfall([
    getDbResults,
    function(dbResults, callback) {
      c.collate(dbResults[0], dbResults[1], callback);
    }
  ], callback);
};
```

```
async.parallel(
  [
    getAndCollate,
    async.apply(c.network, 1)
  ],
  function(err, results) {
    if (err) {
      c.errored();
    } else {
      c.output(results);
    }
  }
);
```

CRUCIBLE 3

PROMISES

```
promises = ids.map(c_p.dbAccess);
```

```
var collatedPromise = Q.all(promises)
  .then(function(dbResults) {
    return c_p.collate(dbResults[0], dbResults[1]);
  });
```

```
Q.all([
    collatedPromise,
    c_p.network(1)
  ])
  .then(
    function(data) {
      c_p.output(data);
    },
    function(err) {
      c_p.errored();
    }
  );
```

CRUCIBLE 3

ICEDCOFFEESCRIPT

```
getAndCollate = (callback) ->
  dbResults = []
  errors = []
  await
    for id, i in ids
      c.dbAccess id, defer errors[i], dbResults[i]

  errors = (err for err in errors when err?)
  if errors.length isnt 0
    callback(errors)
    return

  c.collate dbResults[0], dbResults[1], callback
```

```
await
  c.network 1, defer errNetwork, n1
  getAndCollate defer errCollate, c1

if errNetwork? or errCollate?
  c.errored()
else
  c.output c1, n1
```

WINNER?

VANILLA JS

- You saw that last one
- You will experience that pain on a small scale every day.
- Every. Day.

VANILLA JS

THUMBS DOWN

ASYNC.JS

- It's "close to the metal"
- Stays out of the way when you don't need it
- Powerful abstractions for all common use cases

ASYNC.JS

THUMBS UP

PROMISES

- Requires all tools to support Promises
- Other people can implement Promises poorly and you will pay for it
- But, it's powerful and usually elegant

PROMISES

THUMBS DOWN

ICEDCOFFEESCRIPT

- Very elegant
- Not powerful enough
- Build toolchain problems

ICEDCOFFEESCRIPT

THUMBS DOWN

THANKS!

FOLLOW MY BLOG

technotes.iangreenleaf.com

BUY MY BOOK

bit.ly/coffeescriptappdev

HIRE ME

ian@iangreenleaf.com