ASYNC TESTING

Wait for it…

BY THE END OF THIS LECTURE...

- What is the issue presented by asynchronicity in tests?
- What are two ways to handle async in tests?

A SYNCHRONOUS TEST

```
describe('Add', function() {
  it('works on integers', function() {
    expect(add(1, 2)).to.equal(3);
  });
});
```

How does this work again? 🤔

If `add(1, 2)` does not output 3, the `expect` statement throws an error and the test fails.

If we reach the end of the function without throwing an error, the test passes.

ASYNCHRONOUS TEST?

```
describe('fs.readFile', function() {
  it('reads file contents', function() {
    fs.readFile('file1.txt', function(err, data) {
      if (err) throw err;
      expect(data.toString()).to.equal('Hello!');
    });
  });
});
```

What's wrong with this? 🤔

ASYNCHRONOUS TEST?

```
describe('fs.readFile', function() {
  it('reads file contents', function() {
    fs.readFile('file1.txt', function(err, data) {
      if (err) throw err;
      expect(data.toString()).to.equal('Hello!');
    });
  });
});
```

If we reach the end of this function without throwing, the test passes.

ASYNCHRONOUS TEST?

```
describe('fs.readFile', function() {
  it('reads file contents', function() {
    fs.readFile('file1.txt', function(err, data) {
      if (err) throw err;
      expect(data.toString()).to.equal('Hello!');
    });
  });
});
```

If we reach the end of this function without throwing, the test passes…

… but this function with the assertion *never even runs* before that function completes!

ASYNCHRONOUS TEST?

```
describe('fs.readFile', function() {
  it('reads file contents', function() {
    fs.readFile('file1.txt', function(err, data) {
      if (err) throw err;
      expect(data.toString()).to.equal('Hello!');
    });
  });
});
```

This spec will *always pass!* 

How do I tell Mocha that this test

ASYNCHRONOUS TEST?

```
describe('fs.readFile', function() {
  it('reads file contents', function() {
    fs.readFile('file1.txt', function(err, data) {
      if (err) throw err;
      expect(data.toString()).to.equal('Hello!');
    });
  });
});
```

ASYNCHRONOUS TEST?

How do I tell Mocha that this test isn't "done" at this point

```
describe('fs.readFile', function() {
  it('reads file contents', function() {
    fs.readFile('file1.txt', function(err, data) {
      if (err) throw err;
      expect(data.toString()).to.equal('Hello!');
    });
  });
});
```

ASYNCHRONOUS TEST?

How do I tell Mocha that this test isn't "done" at this point

but instead, at…

```
describe('fs.readFile', function() {
  it('reads file contents', function() {
    fs.readFile('file1.txt', function(err, data) {
      if (err) throw err;
      expect(data.toString()).to.equal('Hello!');
    });
  });
});
```

How do I tell Mocha that this test isn't "done" at this point

ASYNCHRONOUS TEST?

but instead, at…

```
describe('fs.readFile', function() {
  it('reads file contents', function() {
    fs.readFile('file1.txt', function(err, data) {
      if (err) throw err;
      expect(data.toString()).to.equal('Hello!');
    });
  });
});
```

… this point…

ASYNCHRONOUS TEST?

How do I tell Mocha that this test isn't "done" at this point

but instead, at…

```
describe('fs.readFile', function() {
  it('reads file contents', function() {
    fs.readFile('file1.txt', function(err, data) {
      if (err) throw err;
      expect(data.toString()).to.equal('Hello!');
    });
  });
});
```

… this point…

… or this point?

APPROACH 1: THE “DONE” CALLBACK

ASYNCHRONOUS TEST

```
describe('fs.readFile', function() {
  it('reads file contents', function(done) {
    fs.readFile('file1.txt', function(err, data) {
      if (err) done(err);
      expect(data.toString()).to.equal('Hello');
      done();
    });
  });
});
```

Any block that needs to do async stuff may be defined with a done **callback.**

In this case, Mocha will not consider the spec complete until you invoke done

ASYNCHRONOUS TEST

```
describe('fs.readFile', function() {
  it('reads file contents', function(done) {
    fs.readFile('file1.txt', function(err, data) {
      if (err) done(err);
      expect(data.toString()).to.equal('Hello');
      done();
    });
  });
});
```

Invoking done **with an argument signifies to Mocha that something went wrong (like** next **in Express)**

Otherwise, the test is done **once expectations are made.**

APPROACH 2: RETURN YOUR PROMISES

RETURN A PROMISE

- Manually managing all the places to call done is a bit cumbersome
- Fear not! Mocha supports promises!
- If you return a promise in an it (or before/after/etc.) block, Mocha will know to wait for async operations to complete

RETURN A PROMISE

```
describe('promisifiedReadFile', function() {
  it('reads file contents', function() {
    return readFileAsync('file1.txt')
    .then(data => {
      expect(data.toString()).to.equal('Hello!');
    });
  });
});
```


You can return literally return a promise (and do your assertions in a .then block)…

RETURN A PROMISE

```
describe('promisifiedReadFile', function() {
  it('reads file contents', async function() {
    const data = await readFileAsync('file1.txt')
    expect(data.toString()).to.equal('Hello!');
  });
});
```

⚠️🚨 *Notice that there is no try…catch block!* 🚨⚠️

If expect throws, catch would capture - and mostly likely resolve - that error. This is *not* what we want. Leave it, mocha will handle it!

… Or use async/await. Remember that an async function ALWAYS returns a promise, so you don't explicitly need to return anything.

RECAP

- **What is the issue presented by asynchronicity in tests?**
  - Async callbacks will not execute until after the functions in the it blocks complete execution. The spec will have already “passed” by the time we reach expectations inside an async callback.
- **What are two ways to handle async in tests?**
  - The done callback (must call done when appropriate)
  - Returning your promise (less trouble)
  - Never both (mocha will be confused!)

Lab link: https://learn.fullstackacademy.com/workshop/5a6f85d4b9d04700047d99e8/landing