

# SPIES, MOCKS, AND STUBS

*Fake it 'til you make it*

# PREMISE

- **Pure functions are easy to test**
  - Example: addition
    - Input (1, 2, 3) => reliable output (6), the same every time
- **Non-pure functions are trickier**
  - Example 1: readFile
    - Input (filename) => output depends on file contents, which can change even if the filename does not
  - Example 2: func depends on other funcs (e.g. randomized output)
- **We want isolation!**

Pure: no side effects (doesn't talk to the outside world, ajax...), and deterministic (same output for same input, reliably)

# SPIES

# SPIES

- **An object that “spies” on a function to record metadata**
  - Whether the function was called
  - How many times it was called
  - Arguments it was called with
  - etc...
- **Can also be a dummy function to pass around**
  - Output of this function doesn't matter, we only care about how it's being used by another function (e.g. callback with right args)

## SPIES - ANONYMOUS FUNCTION

```
function testFunc(callback) {  
  callback();  
}  
  
describe('testFunc function', () => {  
  it('should call the callback', () => {  
    let mySpy = sinon.spy(); // anonymous spy  
  
    testFunc(mySpy);  
  
    expect(mySpy.calledOnce).toEqual(true);  
  });  
});
```

## SPIES - WRAP AN EXISTING METHOD

```
const user = {
  setLocation: function(location) {
    this.location = location;
  }
}

describe('user object: setLocation function', () => {
  it('should be called with location', () => {
    let mySpy = sinon.spy(user, 'setLocation');

    user.setLocation('Rochester');

    expect(mySpy.calledWith('Rochester')).to.equal(true);
  });
});
```

# STUB

# STUBS

- Like a spy... but fake!
- Spies “spy” on a function (calls original/same behavior)
- Stubs *replace* a function entirely, “stubbing” their output
- Use cases:
  - Replacing external requests (e.g. AJAX, db queries) - takes too long and not predictable outcome
  - Testing behavior tracks based on function output (e.g. coin flip: if Math.random gives >0.5, return heads; else tails)



# STUBS

```
let stub = sinon.stub(Math, 'random');  
stub.returns(/* something specific */);
```

# MOCK

# MOCKS

- **Like a stub... but *with more functionality!***
  - Allows you to stub all methods in an object at once
  - Comes with built-in assertions

**“LEAVE IT HOW YOU FOUND IT”**

# RESTORE

- When using mocks and stubs, you are replacing behavior of functions that have other important tasks to get back to once you're done with testing!
- It's important to `restore` things to their original state whenever appropriate

## WHERE TO FIND THESE COOL TOOLS?!

- **Spies/mocks/stubs often come packaged with your testing framework of choice**
  - Each may have slightly different descriptions of “mock”/“stub”/“spy”
  - No need to get too hung up about terminology - consult specific documentation and find the right tool for your task!
- **You can also pick standalone tools (much like how we chose chai for assertions)**
  - Sinon.js is a popular choice for Mocha