

Unit Testing 2

Overview

- Dependency Injection
- Mocking and Stubbing

Learning Objectives

- To be able to explain what *Dependency Injection* is and why we do it.
- To gain experience *Mocking* and *Stubbing* in order to write well tested code.

Re-cap

In the previous session we learned how to write some unit-tests for our `add_two_numbers` function:

```
def add_two_numbers(a, b):  
    return a + b
```

Consider

What happens when our *unit* depends on the outcome of some other piece of code? How can we then test our *unit* in isolation?


```
def get_country_code(key):  
    countries = get_countries() # Dependency  
    return countries[key]
```

What is a Dependency

Our *units* may depend upon other functions, libraries or external services in order to do their job. We call these *dependencies*.

Example dependencies:

- REST API
- MySQL Database
- File Store
- Print / Input / Math etc
- Any more?



In order to test our units in isolation, we need a mechanism with which to replace these dependencies with dummy functions or data that imitate their behaviour.

How do we do that then?

Dependency Injection (DI)

By injecting the dependency, the caller of our function is responsible for providing the `get_countries` logic.

```
                                # Inject get_countries dependency
def get_country_code(key, get_countries):
    countries = get_countries() # Execute dependency
    return countries[key]
```

Which means that

- When we call `get_country_code` in our application, we inject the real `get_countries` function
- When we call `get_country_code` in our test, we inject a fake (*mock*) `get_countries` function

The Real Function

The Mock Function

Wait for it...

The Difference Being

- The real function calls a third-party API to get a list of Country information.
- The mock returns a *stubbed* list with a single item with only the `key:value` pairs used by our code, imitating the real service only as much as needed.

Let's write the test

Some Caveats of DI

- May require restructuring of your code if retro-fitting.
- Tests will be so easy to write you may die of boredom.
- Your colleagues will be envious of you.
- Recruiters will keep blowing up your phone.

Exercise

Instructor to distribute exercise.

Learning Objectives Revisited

- To be able to explain what *Dependency Injection* is and why we do it.
- To gain experience *Mocking* and *Stubbing* in order to write well tested code.

Terms and Definitions Recap

- **Mock**: A piece of *fake* code standing in to replace some *real* code.
- **Stub**: Dummy data serving to replace real data usually returned from an external source.
- **Dependency**: A piece of code relied upon by another piece of code.
- **Dependency Injection**: A Software Development paradigm in which dependencies are passed as inputs into the function/class that invokes them.

Further Reading

- YouTube: [Dependency Injection \(in JavaScript but still a great watch\)](#)
- [Dependency Injection](#)