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]
```

Initiate a discussion to try and answer these questions, don't give them the answer just yet, but get them thinking.

Point out:

- When we run this function, it also runs `get_countries`.
- `get_countries` may also have it's own deps, and thus execute those.
- We don't necessarily know (without looking) what `get_countries` itself is going to depend on.
- If we leave it as it is, our test will also indirectly test the deps, and deps of deps, which is *Integration Testing*.
- We want to test *only* the `get_country_code` function.

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.

Mocks and Stubs are both subcategories of "Fakes" Stub: a dummy piece of code that lets the test run, but you don't care what happens to it - More concerned with dummy data Mock: a dummy piece of code, that lets you VERIFY it is called correctly as part of the test - More concerned with behaviour

```
Sometimes referred to a little interchangably.
```

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

```
def get_countries():
    headers = {'Content-Type': 'application/json'}
    api_url = "https://restcountries.com/v2/all"

    response = requests.get(api_url, headers=headers)

    if response.status_code == 200:
        return json.loads(response.content.decode('utf-8'))
    else:
        return None

def find_country_capital(country_code, get_countries):
    countries = get_countries()

    for country in countries:
        if country['alpha3Code'] == country_code:
            return country['capital']

    return None

print(find_country_capital("GBR", get_countries))
```

```
[
    {"name": "United States of America" , "alpha3Code": "USA", "capital":
"Washington DC"},
    {"name": "United Kingdom", "alpha3Code": "GBR", "capital": "London"},
    {"name": "Uruguay", "alpha3Code": "URY", "capital": "Montevideo"}
]
```

Ask:

- What do we think this code is doing at a high level?
- Why would it be bad to use an external service?

Points:

- Main reason: Control. Using an external service, we depend on data that is out of our control for our test.
- In this particular case the API will return the same data each time, but what if it didn't? For example an API which provides recent news articles.
- Other reasons: Test execution speed, Test parallelisation, Cost to use external API, API Credential management complexity, Service usage limits...

The Mock Function

Wait for it...

```
def mock_get_countries():
    return [
        {"alpha3Code": "GBR", "capital": "London"}
    ]
```

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.

Ask:

- How much data SHOULD we return here?
- Depends what we are trying to test. In many cases, just the single data point we need might be enough
- If the test is to ensure the correct country is chosen from the list then we could add more to help guard against a naive edge-case implementation, e.g. that just returns the first from the list

Let's write the test

```
def test_find_country_capital():
    # Arrange
    expected = 'London'

    def mock_get_countries():
        return [
            {"alpha3Code": "USA", "capital": "Washington DC"},
            {"alpha3Code": "GBR", "capital": "London"},
            {"alpha3Code": "TKM", "capital": "Ashgabat"}
        ]

    # Act
    actual = find_country_capital('GBR', mock_get_countries)

    # Assert
    assert actual == expected

test_find_country_capital()
```

- Explain Dependency Injection as a design pattern. (Also called Dependency Inversion or Inversion of Control) Gives caller control of dependencies. Dependencies are made explicit as they are described in the contract (function or method signature)

- White Box vs Black Box testing
- Unit testing is very strictly a White Box test - we CAN and absolutely do need to look at the internals to write a good test - and we can refactor the internals to make the test easier
- Demonstrate using Postman or similar to inspect the response data from the API and determine what is required for our test case

---

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