# 823G5 Programming in Python

## Aim

To familiarise yourself with

- Unit testing
- Component testing
- Exceptions

## Exercises

Download the `Lab7Exercises` ZIP file. It contains some basic code for you to develop. The code is a set of Python source code file (`.py`). You will need to open a new PyCharm project and import the `.py` files into it. Refer to the Lab 1 instructions if you cannot remember how to do this.

## Generating unit tests

Unit testing can be used to test functions, methods and classes. We create unit test classes that contain test cases.

- You can have as many test cases as you like (a method starting `test` … is a test case). Each test case can have many actual tests.
- Together they form a test suite.
- Use of `setUp()` and `tearDown()` is optional – called after each complete test case (not after every assert…).

A good starting point is to create a unit test class for each significant class in your codebase. You can build up your test suite as your codebase progresses. You then have a useful test suite that you can deploy at any stage in the codebase development to ensure that everything is working as intended.

## Coding challenge 1: Building a unit test for a single class

Open the `challenge1.py` file. It contains a complete class definition for a `ContactsBook`. We need to develop a test class that can run suitable unit tests to ensure that `ContactsBook` is working correctly.

Create a new separate `TestContactsBook` unit test class that performs two test cases. Remember that a test case is associated with a test method, and

each method can perform one or more unit tests. Create a test case method `test_1()` that performs the following unit tests:

- Adds a new contact ‘william’ with email address ‘will@acme.com’, position ‘junior’ and telephone extension 1234, and then checks that the contact has been added successfully.
- Verifies that the number of contacts in the book is now 4.
- Attempts to add ‘william’ again and then checks that it was not added.
- Verifies that the number of contacts in the book is still 4.

Then create another test case method `test_2()` that performs the following unit tests:

- Verifies that rod’s email address is ‘rod@acme.com’.
- Updates rod’s email address to ‘roddy@acme.com’.
- Verifies that rod’s email address is now ‘roddy@acme.com’.
- Verifies that rod can be looked up by name.
- Check that if we try to look up a person whose name is not in the contact book that `search_by_name()` returns `False`.

A suggested answer will be provided on Canvas.

**Coding challenge 2: Component level testing**

Component level testing is where we test several classes or code elements working together. The idea is that these classes provided a significant component of the solution, but not a solution for the whole problem. Think of it as testing of a significant sub system. In some cases, component testing can be used as system level testing.

Open the `challenge2.py` file. It contains 2 classes for an online shopping application; Customer and Item. They work together as a component part of a larger application.

Create a new separate `TestShoppingApplication` class that will provide unit tests for the Customer and Item classes working together.
Create appropriate `setUp()` and `tearDown()` methods to manage 3 `Customer` objects and 3 `Item` objects:

- `c1`: A `Customer` object with name ‘bob and email ‘bob@acme.com’.
- `c2`: A `Customer` object with name ‘alice’ and email ‘alice@acme.com’.
- `c3`: A `Customer` object with name ‘jimmy’ and email ‘jimmy@acme.com’.
- `i1`: An `Item` object that is a ‘small table’ costing £50.
- `i2`: An `Item` object that is a ‘table lamp’ costing £30.

- `i3`: An `Item` object that is a ‘rug’ costing £100.

Create a test case method `test_1()` that creates data as follows:

- `c1` buys an `i1`
- `c1` buys an `i2`
- `c2` buys an `i1`
- `c2` buys an `i3`

The method `test_1()` should then perform the following unit tests:

- Checks that the value of the `c1` basket is £80.
- Checks that the value of the `c2` basket is £150.
- Checks that the value of the `c3` basket is £0.
- Removes `i1` from the `c1` basket correctly.
- Checks that `i3` cannot be removed from the `c1` basket.
- Removes a table lamp by name from the `c1` basket.
- Removes a small table by name from the `c2` basket.
- Checks that the value if the `c1` basket is now £0.
- Checks that the value of the `c2` basket is now £100.

Create a test case method `test_2()` that creates data as follows:

- `c1` buys an `i1`
- `c1` buys an `i2`
- `c2` buys an `i1`
- `c2` buys an `i3`
- `c3` buys an `i3`

The method `test_2()` should then perform the following unit tests:

- Correctly changes the `i3` name to ‘red rug’.
- Checks that the value of the `c1` basket is £80.
- Checks that the value of the `c2` basket is £150.
- Checks that the value of the `c3` basket is £100.
- Correctly applies a discount of 10% to the `c1` basket (making it worth £72).
- Checks that the value of the `c1` basket is now £72.
- Tries to apply a discount of 60% to the `c2` basket but does not succeed.
- Checks that the value of the `c2` basket remains as £150.

An example solution will be made available on Canvas.

**Coding challenge 3: Using Exceptions**

Open up the `challenge3.py` file. It is a version of the online shopping application from challenge 2. This time, there is a `main()` method. The methods `remove_item_by_name()` and `remove_item()` have been rewritten so that they raise a custom `ItemNotPresentError` exception if the item is not present in the customer basket.

You should implement the `ItemNotPresentError` exception so that the exception can be raised correctly.

Rewrite the `discount_basket()` so that it raises a custom exception `InvalidDiscountRateError` if the specified rate is not within the permitted range 10%-50%. Implement the `InvalidDiscountRateError` exception.

An example solution will be made available on Canvas.

Dr. Benjamin Evans
B.D.Evans@sussex.ac.uk