FIT4004 Assignment 3

Automated Unit Testing and Continuous Integration

15% of your final mark

Incremental submission - read the instructions *carefully*!

Parts of this assignments will be completed in pairs

In this assignment, you will create a small module in Python, and test it using `unittest` and `mock`. You will also set up continuous integration, so that testing occurs as you update your source code repository.

Typically, developers are expected to write unit tests for their own code, hence the need to write your own test suite. However, you are *not* going to be marked on the quality of your implementation, merely that you have one to test against. Hopefully, doing comprehensive unit testing and fixing bugs that you find will result in a module that is close to functionally correct.

Groups

Parts of this assignment will be done in groups of two (or, if there are uneven numbers and there is no alternative, groups of three).

The Module (written in pairs)

The description of your module seems quite lengthy. You'll be reassured to know that it's actually lengthier than the code I wrote to implement it!

Your module, entitled `mailmerge`, will provide a simple *mail merge* facility, for automatically generating personalized email messages and sending the personalized messages to a group of recipients.

The mail merge will perform its work by performing *macro substitutions* on a *template*. There are two forms of macros - *scalar macros* and *loop macros .*

A template contains plain text and the two macro forms, which can appear at any point within the text. A scalar substitution is of the form "`$(`*`macro_name`*`)`" where *`macro_name`* is one or more alphanumeric characters that uniquely name the macro. A loop substitution is of the form `$FOR(`*`loop_macro_name`*`, "`*`loop_template`*`")`, where *`loop_macro_name`* is one or more alphanumeric characters that uniquely name the macro (note: loop macro names and scalar macro names share the same namespace), and the template is a plain text of the same form, with the restriction that it cannot contain other loop macros. For simplicity, you can assume it may not contain literal quote (") characters either.

The text to be substituted is defined by a Python dictionary whose keys are macro names. In the case of scalar macros, the value related to the key is a plain text string, which replaces the macro text.

For a loop macro, the key is related to a *list of dictionaries.* For each dictionary in the list, you produce a copy of the template, performing substitutions based on the items in that dictionary.

Note that the scalar substitutions available within the global namespace are not available within a loop macro.

If a macro variable is undefined, a `MacroNotDefined` exception should be thrown.

The module should define a method `fill_template(template, subvars)` where `template` is a template in the above format (as a Python string), and `subvars` is a Python dictionary in the format described. This method should return a Python string.

To make this a little easier to understand, have a look at the following Python code:

```
from mailmerge import *

mysubs, jack, fred = {}, {}, {}
mysubs["DANCER"] = "Ginger Rogers"
mysubs["FILM"] = "Puttin' on My Top Hat"
jack["STEP"] = "Foxtrot"
jack["SCENE"] = "1"
fred["STEP"] = "Rhumba"
fred["SCENE"] = "2"
mysubs["DANCELIST"] = [jack, fred]

mymessage = "$(DANCER) was in $(FILM).  $(DANCER) is a dancer.\
$FOR(DANCELIST,\"They danced the $(STEP) in scene $(SCENE)\")"

print(fill_template(mymessage, mysubs))
```

The output from running this code should be:

```
Ginger Rogers was in Puttin' on My Top Hat.  Ginger Rogers is a dancer.  They
danced the Foxtrot in Scene 1 They danced the Rhumba in scene 2
```

The module should also define a `MailMerge` class, which supports a constructor which takes the following arguments:
`host` - the mail host you will use to send mail.

`username` - the user name you will use to log into your mail host
`password` - the password you use to log into your mail host
`from_address` - the address from which mail will appear.

The class supports a `mailmerge` method, which takes the following arguments (as well as `self`):

`template` - a template as for fill_template
`subject` - the subject for the emails, as a string
`subvarlist` - a list of dictionaries, containing the data to be applied to template to create each email. Each dictionary should contain a key "to", which contains the address to send the mail as a string.

Calling this method uses the mail server to send messages, customized based on the template and the dictionaries, to the recipients specified in each dictionary.

For instance, if you added this code to the code above:

```
mysecondsub = copy.deepcopy(mysubs)
mysecondsub["DANCER"]="Fred Astaire"
mysubs["to"]="fred.nurk@random.org"
mysecondsub["to"] ="gina.g@eurovisionretirementhome.com"

merger = MailMerge("smtp.gmail.com:587","randomid", "notarealpassword",
"from-mail@gmail.com")
merger.mailmerge(mymessage, "test message", [mysubs, mysecondsub])
```

This would send out an email to Fred Nurk and Gina G, one informing Fred Nurk about Ginger Rogers' career, another informing Gina G about Fred Astaire's activities (but otherwise identical).

Email functionality is very easy in Python, with the use of `smtplib` and `email` packages in Python (go to the Python docs). You can use Gmail's SMTP server (see https://support.google.com/a/answer/176600?hl=en) but it's also pretty trivial to set up your own local mail server on a Linux box (or virtual machine).

My sample solution is about 50 lines of code (excluding comments, and without exception-throwing implemented). If your code is orders of magnitude longer than this, you are overthinking your code.

YOU WILL NOT BE MARKED ON THE ELEGANCE, EFFICIENCY, MAINTAINABILITY, OR ANY OTHER PROPERTY OF THE MODULE OTHER THAN FUNCTIONAL CORRECTNESS AND BASIC READABILITY. The focus is on the testing of the code, not the code itself.

Your test plan

Your test plan should be simple. You only need to test the functional correctness of the module and run the tests using the unittest module, performing continuous integration using drone.io.

You must "mock out" anything that accesses a network using the `mock` module.

If the contents of the calls to network access methods are things that you should check to ensure the module is working correctly, you should use the facilities within `mock` to check their correctness. However, you should do this *after* getting your tests working - this may be time consuming and is less important than getting a good set of tests up and running.

The strategy you use to select your test cases is up to you. You should explicitly document your test case selection strategy in a plain text file called "`test-strategy.txt`" and add it to the repository.

If you want to measure statement or branch coverage as part of your test strategy, there is a standard python tool called `coverage.py` that can help you achieve this:

http://coverage.readthedocs.io/en/coverage-4.0.3/

Unit Tests (pairs)

Based on your test strategy, devise a test suite for the module.

You should use your unit tests to fix bugs in your module.

Repository and Continuous Integration (pairs)

Work on this project should all be kept in a BitBucket repository - create a new project for this Assignment, give the teaching staff (Robyn McNamara and Robert Merkel) full access, then transfer ownership of the repository to Robert following these instructions https://confluence.atlassian.com/display/BITBUCKET/Change+or+transfer+repository+ownershi p. This is to get continuous integration to work!

I have an account on a cloud-based continuous integration service called "drone.io". You should all register - when you create your repositories, let me know and I will create a drone.io project linked to your repository and give you admin access to that project.

You should then follow drone.io's instructions to configure your unit tests to run on drone.io's servers each time you update your BitBucket repository.

Note that drone.io only supports a restricted number of Python versions (2.7, 3.3 and 3.4, from memory). Feel free to choose whichever Python version you like from the supported versions.;

Test analysis (individual)

As in previous assignments, after completing the testing process, write a short report (no longer than about two pages), about the test process, including:

- What bugs did you find with your unit tests?
- Did you find the coverage tool useful to assess the adequacy of your tests?
- Do you think it works better for the individual writing a specific piece of code to write the tests, or for somebody else to do it?
- How long did writing mock code take? Was it a major component of the overall effort?
- What would you do differently next time if faced with a similar task?

Marking criteria

- Completion of module according to spec (minor consideration - remember, you are not being marked on the quality of the implementation).
- Reasonableness of test case selection strategy.
- Quality of test cases (including quality of test case code).
- Effectiveness of mock usage.
- Appropriate usage of CI system.
- Quality of test analysis
- Readability of test analysis.

Submission

Submission will be performed by uploading all items to your repositories.

Your completed module and unit tests should be submitted by Thursday 26th May at 5PM.

Your test analysis report should be uploaded by Friday 27th May at 11:55 PM (that's as late as I can make it).

Plagiarism

You're in fourth year. You know the drill by now.

Special consideration

If a student faces exceptional circumstances (serious illness or injury, family emergency etc) that prevent them completing the assignment, they may apply for special consideration according to the policy and procedure on the Faculty website:

http://www.infotech.monash.edu.au/resources/student/equity/special-consideration.html

Other late submissions

A penalty of 10% per working day will apply to submissions after the deadline.

Assignment forum

You can ask questions on the discussion forum, which the lecturer will monitor and respond to daily (and at least once on weekends). All students should monitor this forum - I may clarify aspects of the assignment on the forum and any such clarifications are considered "official". You may of course also email the lecturer, or arrange to see them in person if you prefer.