UCL DEPARTMENT OF MEDICAL PHYSICS AND BIOMEDICAL ENGINEERING

Research Software Engineering With Python

The Alchemist's Laboratory - a package for any alchemist!

Module Code:	MPHY0021
Module Title:	Research Software Engineering With Python
Lecturer(s):	Dr Matt Clarkson Dr Anastasis Georgoulas Dr David Pérez-Suárez
Coursework Title:	The Alchemist's Laboratory - a package for any alchemist!
Date Handed out:	November 1st, 2018
Coursework Deadline:	January 4th, 2019
Submission Id:	258203

Total mark:
20.3/*25*

Description: This assignment asked to refactor existing code and package it in a form that can be tested, installed and accessed by other users. The code to actually solve the problem was already given, but as roughly sketched out code in a notebook. Your job consisted in converting the code into **a formally structured package**, with **unit tests**, **docstrings**, a **command line interface**, using proper **object oriented structures** and demonstrating your ability to **use git version control**. This exercise has been semi-automatically marked.

Marking legend:

Title of the grading section *points received*/*total*

General notes about this particular section

section being marked — *auto* *manual* **total**

Feedback about this section
Comments here don't necessarily subtract points
Automatically graded marks either worked or not. Noted if they had to be run manually.

Code in `laboratory.py`, implementing the full experiment reaction *3.00* / *5*

Which works	*1*	*0*	**1.00**
Cleanly laid out and formatted - PEP8 *Does `pycodestyle` produce errors?*	*1*	*0*	**1.00**
Defining the class Laboratory (and maybe Substance) with a valid object-oriented structure *Not using the object on functions like `update_shelves` or `do_a_reaction`.* *No updated the state of the laboratory.* *Docstrings would be helpful.*	*0*	*0*	**0.00**
Breaking down the solution sensibly into subunits	*0*	*1*	**1.00**
Structured so that it could be used as a base for other type of reactions `can_react` *is fixed on the laboratory. An external class either related with the substance or independently would provide this desired requirement.*	*0*	*0*	**0.00**

Command line entry point

5.00 / *5*

Accepting a laboratory definition text file as input *Does* `abracadabra` *exist? Does it accept an input* `yaml` *file?*	*1*	*0*	**1.00**
With an optional parameter to output the number of reactions *is* `reactions` *accepted and produce the right output?*	*1*	*0*	**1.00**
Which prints the result to standard out *Is the output properly formatted as a yaml file?*	*1*	*0*	**1.00**
Which correctly uses the Argparse library	*0*	*1*	**1.00**
Which is itself cleanly laid out and formatted *There's no need to unpack into a list the result of the methods.*	*0*	*1*	**1.00**

---

setup.py file

4.00 / *5*

This section is fully marked automatically.

Which could be used to 'pip install' the project `pip install .` *didn't fail*	*1*	*0*	**1.00**
With appropriate metadata, including version number and author `pip show package_name` *displays such information.*	*1*	*0*	**1.00**
Which packages code (but not tests), correctly.	*0*	*0*	**0.00**
Which specifies library dependencies	*1*	*0*	**1.00**
Which points to the entry point function	*1*	*0*	**1.00**

Three other metadata files

2.50 / *3*

1 point per file present. Marks removed if the content is not meaningful.

Who did it, how to reference it, who can copy it *The content of the readme file is not relevant to this library.*	*1*	*1.5*	**2.50**

Unit tests

3.80 / *5*

1 point if `pytest` run automatically without errors (distributed as 0.2 on each subsection). Maximum mark per subsection is 1 point (0.2 automatically and 0.8 manually).

Which test some obvious cases *Not unit tested the different methods on the Laboratory*	*0.2*	*0.4*	**0.60**
Which correctly handle random selections *Not tested*	*0.2*	*0.0*	**0.20**
Which test how the code fails when invoked incorrectly	*0.2*	*0.8*	**1.00**
Which use a fixture file or other approach to avoid overly repetitive test code *DRY fixture only used on the first test.*	*0.2*	*0.8*	**1.00**
Which are themselves cleanly laid out code *Though command should have been imported from ..command*	*0.2*	*0.8*	**1.00**

Version control *2.00* / *2*

0.5 point in total if `git` was used in the project (distributed as 0.25 on each subsection). Maximum mark per subsection is 1 point (0.25 automatically and 0.75 manually).

Sensible commit sizes	*0.25*	*0.75*	**1.00**
added binaries files to the repository but then removed at the end.			
Appropriate commit comments	*0.25*	*0.75*	**1.00**