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:	258229

Total mark:
16.75/*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 *2.50* / *5*

	auto	manual	total
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 *Docstrings would be helpful.* *Not using the object on functions like `update_shelves` or `do_a_reaction`.* *`run_full_experiment` doesn't update the estate of the object.*	*0*	*0*	**0.00**
Breaking down the solution sensibly into subunits *Why is pytest needed for the laboratory?* *Error or warning messages should not use `prints`* *If a method `prints` should return nothing.* *Variables like `k1`, `k2`, ... are meaningless.*	*0*	*0.5*	**0.50**
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

1.50 / *5*

Accepting a laboratory definition text file as input	*0*	*0*	**0.00**
Does `abracadabra` *exist? Does it accept an input* `yaml` *file?*			
With an optional parameter to output the number of reactions	*0*	*0*	**0.00**
is `reactions` *accepted and produce the right output?*			
Which prints the result to standard out	*0*	*0*	**0.00**
Is the output properly formatted as a yaml file?			
Which correctly uses the Argparse library	*0*	*1*	**1.00**
Which is itself cleanly laid out and formatted	*0*	*0.5*	**0.50**
Missing help for the arguments. How does a user know what the expected format of the `filename` *is?*			

setup.py file

5.00 / *5*

This section is fully marked automatically.

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

Three other metadata files

3.00 / *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	*1*	*2*	**3.00**
Readme only mentions the usage of the entry point but not of the library nor its installation.			

Unit tests

3.00 / *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	*0.2*	*0.0*	**0.20**
Not tested anything, just the constructor.			
Which correctly handle random selections	*0.2*	*0.0*	**0.20**
Wrong use of 'a == b or c or d', here `a` *is not been compared with all but only with the first one,* `b`*. If that's False, then if* `c` *or* `d` *is not Falsy (False, and empty list, an empty string,...) then it gives you True. Being therefore a false positive. Besides, this is not testing the solution, but the constructor!!*			
Which test how the code fails when invoked incorrectly	*0.2*	*0.8*	**1.00**
Cases tested, but not the message produced.			
Which use a fixture file or other approach to avoid overly repetitive test code	*0.2*	*0.4*	**0.60**
Only using fixture files as to load data, not to make code DRYer.			
Which are themselves cleanly laid out code	*0.2*	*0.8*	**1.00**

Version control *1.75* /*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 *added binaries files to the repository.*	*0.25*	*0.5*	**0.75**
Appropriate commit comments	*0.25*	*0.75*	**1.00**