TDD Hackathon

Introduction

During the hackathon, you will be creating your initial assessment coding application with a TDD approach. Below you will find a number of "tickets" which you will need to complete in order to have delivered the application.
For each coding addition you make, you will need to write an associated test which will be executed via **pytest**. Each change will require a new git branch and you must also create a Pull Request for each completed ticket to merge in the associated branch.
For testing, the aim is to reach **80% overall code coverage** (as reported by pytest).

Preparation

You must have the following in place:

- A GitHub repository named: **multiverse-devops-assessment-2**
- An installation of **Docker** or **Python3 + Pip**
- An installation **pytest** with the coverage extension

Add the pytest configuration file

Create a file named **pytest.ini** in the root directory of your repository and add the following content:

```
[pytest]
python_files = tests.py
addopts=--tb=native --cov=calc --cov=helpers --cov-report term-missing
```

Don't forget to commit this file to your repository!

Installing pytest (Docker)

1. Create a Dockerfile

```
FROM python:3-slim
RUN pip install -U pytest pytest-cov
```

2. Build the docker container image

```
docker build . -t mvpython
```

3. Test the image

```
docker run -it --rm mvpython pip list
```

Your output should appear similar to the following:

```
Package    Version
---------- -------
attrs      22.2.0
coverage   7.0.5
iniconfig  2.0.0
packaging  23.0
pip        22.3.1
pluggy     1.0.0
pytest     7.2.1
pytest-cov 4.0.0
setuptools 65.5.0
wheel      0.38.4
```

4. Executing your tests

After adding your tests, you will execute them with the following command:

```
docker run -it --rm -v "$PWD":/root -w /root mvpython pytest
```

Installing pytest (non-Docker)

pytest (and the coverage extension) is installed via Pip with the following command:

```
pip install -U pytest pytest-cov
```

You can verify the installation is successful by executing: `pip list`.
In the output you should see an entry for **both** `pytest` and `pytest-cov`.

To execute your tests, simply run the command: `pytest`.

The CSV file

Whilst you create your application, it is recommended you use the same CSV file in order to validate consistency and properly perform your Peer Reviews.

Create a new file called results.csv in the root of your repository and add the following content:

```
user_id,first_name,last_name,answer_1,answer_2,answer_3
1,Charissa,Clark,yes,c,7
2,richard,McKinney,yes,b,7
,,,,,
3,patience,reeves,yes,b,9
4,Harding,Estrada,no,b,14
5,India,Gentry,yes,c,7
6,Abra,Sheppard,yes,b,6
6,Abra,Sheppard,no,a,8
7,Bryar,cooley,yes,a,11
8,Diana,Cameron,yes,b,9
9,Alexander,Herring,no,b,4
10,Graiden,Cannon,no,b,13
11,Uma,Glass,yes,a,2
12,Brittany,Weeks,yes,b,8
13,Roth,Stout,yes,c,10
14,Amos,Daniel,yes,a,5
,,,,,
15,Caesar,Rivers,yes,b,7
16,Eugenia,Nichols,yes,b,6
17,dieter,alvarado,yes,b,6
17,Dieter,Alvarado,no,c,7
17,Dieter,Alvarado,yes,a,9
18,Roary,Frank,yes,c,7
19,Ulric,Hensley,no,b,9
20,Felicia,Wilkins,yes,b,8
```

Tickets

Ticket #1	Read a CSV file
Description	In your **input** script, create a function that will read data from a CSV file.
Objectives	● The **results.csv** data file can be successfully processed into an array. ● Each line of the file is read into a new array item. ● The file read method must be in a sub-module.

Ticket #2	Remove duplicate entries
Description	Add functionality to your **input** script to ignore or remove any duplicate entries from the input data. Duplicates are based on the User Id field.
Objectives	● A final array is created with duplicate entries removed. ● Where duplicates are found, the first entry is retained.

Ticket #3	Ignore empty lines
Description	Update your **input** script to ignore any empty lines found when reading in the input data file.
Objectives	● A final array is created with any empty lines omitted.

Ticket #4	Capitalise user name fields
Description	Add functionality to your **input** script to automatically capitalise the **first_name** and **last_name** fields found in the input data.
Objectives	● All names are capitalised in all data entries.

Ticket #5	Validate the responses to answer 3
Description	Update your **input** script to validate the responses to the third answer field. This answer must have a numeric value between 1 and 10. Any rows with an invalid value are ignored.
Objectives	● A final array is created with the input data excluding any rows where answer 3 is invalid. ● No answer 3 values will be outside the range of 1 to 10.

Ticket #6	Output the cleansed result data to a new file
Description	Add functionality to your **input** script to output the cleansed data to a new CSV file.
Objectives	● A new file is created called **clean_results.csv**. ● The file is recreated on each execution. ● No invalid or unformatted data is present in the new file.

Ticket #7	Create an output script
Description	A new **output** script will be created which reads in the **clean_results.csv** CSV file and outputs the results to the command line, row by row.
Objectives	● The script uses the existing sub-module to read the CSV file. ● The printed output will contain all row data and an appropriate header. ● **Stretch:** The printed output will be formatted with fixed length strings.