Aufgaben Mocking in Python

1. Vorbereitung

Bevor wir mit den Aufgaben starten können, müssen wir zunächst die Infrastruktur aufbauen:

1. Dein Kollege hat dieses Projekt mit der Python-Version 3.8.3 gelöst. Es wäre also gut, wenn du diese Version auch hast.
2. Lade dir dann das git-Verzeichnis git@git.cc-demo.com:tutorials/python/Python_Tutorial.git herunter.
   Gehe hierfür zu VCS → Get from Version Control und gebe obigen URL ein.
3. Öffne den Ordner Teil_5_Mocking/Aufgaben.
4. Installiere Pytest (pip install pytest) und Mocking (pip install mock).
5. Es kann sein, dass du außerdem das Package Pandas importieren musst. Gehe in PyCharm Prefereces → Project: Python_Tutorial → Project Interpreter und klicke auf das +, nun kannst du Pandas hinzufügen.
6. Jetzt kannst du mit der Bearbeitung starten

2. Beschreibung

Sie sollen bei der Entwicklung einer neuen Krankenhaussoftware mithelfen. Im Ordner scripts finden Sie die dazugehörige Datei. Es soll ein Patienten-Verwaltungs-Portal entwickelt werden, in dem Ärzte Patienteninformationen verwalten können.
Das Krankenhaus hat den Datensatz Aufgaben/data/BigMedikamente.csv bereitgestellt. Glücklicherweise hat ein Mitarbeiter schon Vorarbeit geleistet und einige Funktionen geschrieben und mit Tests versehen. Allerdings fehlt ein Datensatz, sodass wichtige Funktionen noch nicht implementiert bzw. genutzt werden können.
Dir wurde aufgetragen, mit deinem Wissen aus dem Tutorial Mocking den unvollständigen Code zu testen und ggfs. Dummy-Implementierungen zu verwenden.

Bitte achte bei der Bearbeitung auch auf unsere Codingstandards und insbesondere auf Folgendes:

- Jede Funktion benötigt eine vollständige Beschreibung.
- Jede Funktion benötigt die passenden Tests.
- Jede Funktion muss verwendet werden.
- Jeder Test benötigt eine ausführliche Beschreibung. Achte dabei auf die Standards, die wir in Teil_4_Testing festgelegt haben.

3. Aufgabe

1. Schauen Sie sich den bisher ausgearbeiteten Code genau an.
2. Implementieren Sie den Test **test/Unit/test.TestPatient_Portal.test_str_functional_happypath**. Mit diesem Test soll die Funktionalität der Funktion Patienten_Portal/Patient.__str__() überprüft werden. Leider ist die Funktion Patienten_Portal/Patient.determine_disease() noch nicht implementiert und wir können nicht einsehen, welche Krankheit der Patient besitzt. Nutzen Sie funktionales Patching, um die fehlende Funktion zu mocken und mit einem von Ihnen gewählten return_value zu ersetzen. Überprüfen Sie, ob die Ausgaben von Patienten_Portal/Patient.__str__() korrekt sind.
   Gehen Sie alle vier möglichen Krankheiten mit einem von Ihnen gewählten Beispiel durch.

Tipp: Der Test ist ein Unittest, der über die bereits vorhandene main-Methode ausgeführt wird. Bzgl. Patching bietet sich an, sich an Folie 12 der Präsentation zu orientieren.
beispielhafte Ausgabe: "Patient_123456 hatte am 01.01.2020 einen Termin bei Dr. C. Der Patient gab an, Schmerzen am Knie der Stufe 2 zu verspüren."

3. Prüfen Sie anschließend, ob die gepatchte Funktion jeweils genau einmal aufgerufen wurde.
4. Schreiben Sie einen neuen Test **test/Unit/test_add_patient_2_happypath** , bei dem du patient_1, patient_2, patient_3 wie in test_add_patient_happypath erstellst. Ändere bei patient_2 den Namen des Doktors zu „Dr. E“ und korrigiere die patient_number von patient_3.
   Implementieren Sie nun den Test so, dass die Fehlermeldung wrong_patient_date_error bei patient_1 und patient_3 geworfen wird, wenn der jeweilige Patient in die patient_list hinzugefügt wird. Beim Hinzufügen von patient_2 soll kein Fehler geworfen werden . Die Behandlungsdaten der Patienten dürfen nicht geändert werden.
   Tipp: Nutze Mock.side_effect, um die Rückgabewerte des Moduls datetime zu kontrollieren.