Question 2 - Writing test cases for Question 1 [20 marks]

Consider the programming problem description of Question 1 - Student Information Center. You will need to write tests for the following functions of your program:

1. Function for adding a course (function: **add\_course**)
2. Function for adding a student (function: **add\_student**)
3. Function for adding a teacher (function: **add\_teacher**)
4. Function for listing all students (function: **list\_students**)
5. Function for listing all teachers (function: **list\_teachers**)
6. Function to search for a student by their name or student ID (function: **search\_student**)
7. Function for listing the teachers and their courses for a student (function: **list\_teachers\_courses**)
8. Function to show the GPA of a course (function: **course\_gpa**)
9. Additionally, the recursive function to calculate the GPA for students (function: **student\_gpa**)

Just to make it clear, you will answer part A of this question in test\_plan.md and part B of this question in test.py.

**Part A** [10 marks]

You can start by describing the steps to test a program solution based on the question description. You will plan your test cases in test\_plan.md. You are expected to explain your test cases and demonstrate how you will implement them.  
  
You are testing all the functionality of functions for each menu option of Question 1. You must have at least 6 test cases for each function: 2 positive test cases, 2 negative test cases, and 2 edge cases. Your answer must include the test objectives and example test cases (2 positive cases, 2 negative cases, and 2 edge cases for each function). The test case descriptions must include details of the inputs and expected output. The file test\_plan.md will be written in the Question 1 workspace.

Note that there might be a chance that you can not add any negative or edge cases for some functions, in those cases simply exclude them. As an example, if a function can only be tested with positive cases, you'll only have to write 2 positive test cases. However, it needs to be justified why these cases cannot be done.

Note that this does part does not require any code, and ideally should not.

**Part B** [10 marks]

There is one function for which you must implement the test cases, requiring code: is\_valid\_credit\_hour in the file test.py. The test program will be run against various implementations of the functions: some correct implementations and many incorrect implementations. This is to ensure your test program can catch a wide range of bugs. You are **not** required to implement the function in this part, just to write tests in test.py (you implemented this function in Question 1).

The test cases for this function should consider all functionalities of your code, and consider the 6 positive, negative, and edge cases from Part A. You must implement at least 2 positive test cases, 2 negative test cases, and 2 edge cases. The file test.py will be implemented in the Question 1 workspace.

The test program must print **one** line to the terminal in the following format: <function\_name> has <pass\_status>. If the function being tested failed one or more of your tests, the test suite should print <function\_name> has failed. **once only**. If the function being tested passes all tests, the test suite should print <function\_name> has passed. **once only**.

The correct implementation must be identified by your test suite in order to receive marks for identifying the incorrect implementations. The marking script first calls your test suite on the correct version and then an incorrect version.

Example usages of test.py:

$ python test.py

is\_valid\_credit\_hour has passed.

$

$ python test.py

is\_valid\_credit\_hour has failed.

$

Note: You will **not** be able to run the file, but you can check your implementation using the test cases

**Function specifications**

Ensure you are testing the function, not implementing it

is\_valid\_credit\_hour(credit\_hour: str) -> bool

is\_valid\_credit\_hour() accepts one positional argument credit\_hour and returns a boolean. The function returns True if the credit hour is valid and False if the credit hour is invalid.

**Validation Rule:**

* The credit hour for each course should be a string representation of an integer between 1 and 5 inclusive.