Connor Loughlin, Connor Armstrong, Matthew Ibrahim

Unit Testing Overview and Explanation

Note: We were not able to locate a place where another group gave us peer reviews. A member of our group did submit a review for another team's cases, but we never received one, or at least we were never able to find it.

Function 1

To design the tests for function 1, we started by separating out each of the input variables. Each of the input variables needed to fit within certain parameters in order to ensure that they were viable, if any of the function inputs did not adhere to the parameters the function needed to fail gracefully. In order to test for this we first created equivalence classes for all of the possible valid and invalid parameters. This included tests for invalid input types and tests for boundary values. We checked for the following rules…

For the credit card number we verified that it was…
- The correct length
- An integer
- And Luhn algorithm compliant

For the Identity card number we verified it was…
- 9 characters
- A string
- Nif algorithm compliant

For the name we verified that it was…
- A string
- Greater than 10 characters and less than 50
- Contained both a name and a surname separated by a space character

For room type we verified that it was …
- Either "single", "double", or "triple"
- A string

For the arrival date we verified that it…
- Was formatted dd/mm/yyyy
- Was a string
- That the day value was between 1 and 31 (inclusive) and the month value was between 1 and 12 (inclusive)

For the num days we verified that it was…

- An integer
- Greater than or equal to 1 and less than or equal to 12
-

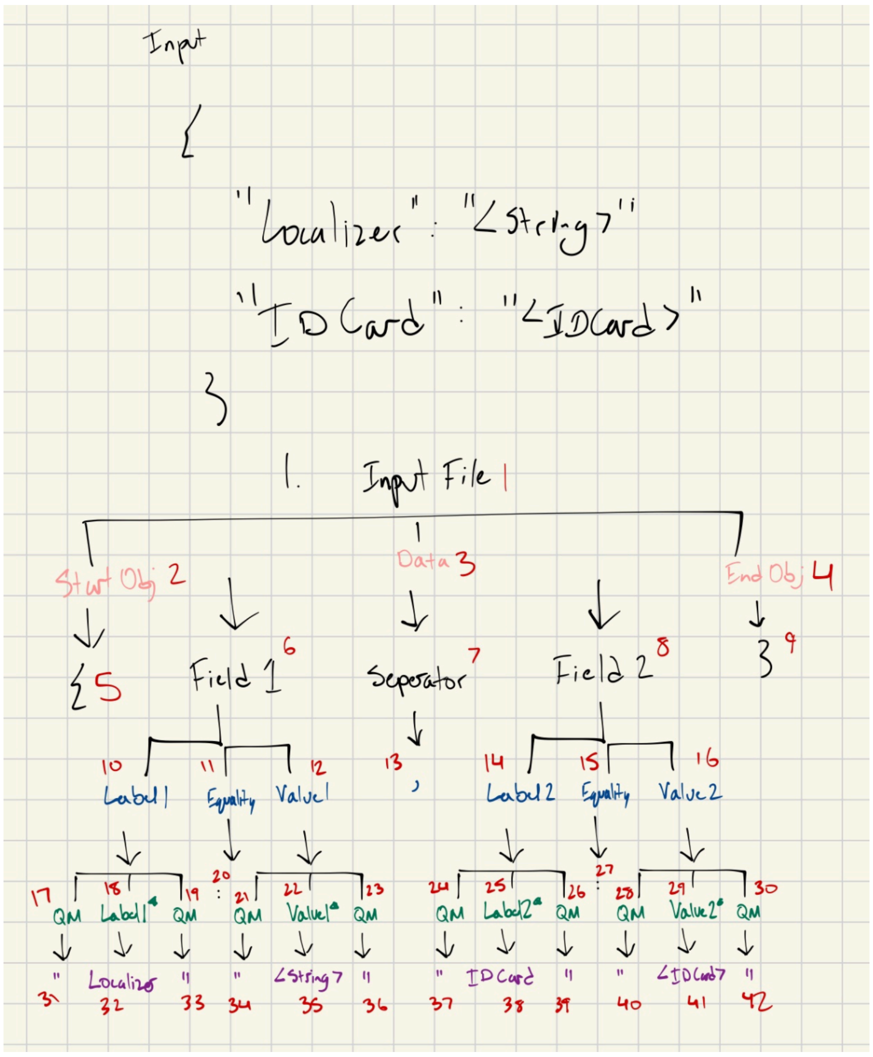
Our full equivalence classes and test parameters can be found on the attached excel file

In between each test case we made use of the setupclass method in unittest in order to avoid the issue of having multiple reservation under the same name in the file. The setupclass method deleted the contents of the reservation file.

We also include a test that attempts to make two reservations with the same name in order to ensure that the second one fails.

Function 2

Checking into the Hotel and guest_arrival

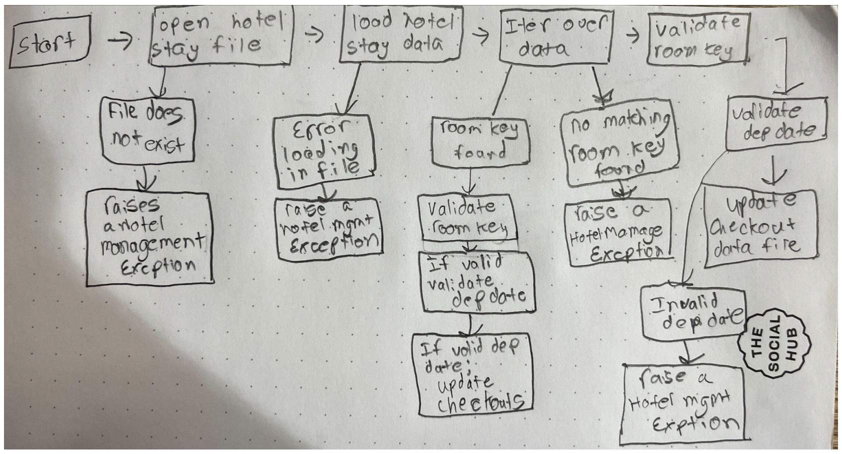


Test cases 1-69 for function 2 (the hotel_stay_tests) are based on this grammar and derivation tree. For each node, we tested with the applicable options of valid, deletion, duplication, and modification to ensure that the JSON input to guest_arrival could only be interpreted if it was perfectly valid. For the terminal nodes, this means we did all three of deletion, duplication, and modification. For nonterminal nodes, we did deletion and duplication, as instructed by the theoretical content. These test cases ensure that every possible exception raised during the process of a guest arrival is proven to work within the scope of the grammar tree.

Test cases 70-77 are supplemental to this grammar tree, and we used the ECBV methodology to ensure all key cases were covered for F2. This meant we tried too long, too short, and non-matching localizers, while also trying a matching localizer that didn't have the correct ID value. We also implemented 2 invalid tests to ensure that the functionality of arriving at the right time was correct as well, one too early and one too late.

Taken together, these tests thoroughly test each line of F2 and make sure that every possible option has been proven to work before this is pushed out to a theoretical production scenario.

Function 3:



Start
open hotel stay file
load hotel stay data
Iter over data
validate room key
File does not exist
raises a hotel management Exception
Error loading in file
raise a hotel mgmt Exception
room key found
Validate room key
If valid validate dep date
If valid dep date; update checkouts
no matching room key found
raise a HotelManage Exception
validate dep date
update checkout data file
Invalid dep date
raise a Hotel mgmt Exption

Function 3, guest_checkout, handled the checking out operations of the Hotel Management system. Additionally, it also produced an output json file that took track of each person checking out.

The test case process followed this general line of reasoning and rules:

1. Valid Guest Checkout:
Input Parameters: Valid room key and Valid departure date

Expected Outcome: The function should successfully process the checkout and update the checkout data file. It should return True.

2. Invalid Room Key:
Input Parameters: Invalid room key.
Expected Outcome: The function should raise a HotelManagementException with an error message indicating that the room key is not registered.

3. Valid Departure Date:
Input Parameters:Valid Departure Date.
Expected Outcome: The function should output a True statement in order to indicate a valid departure date that exists in the larger file

4. Valid Room Key:
Input Parameters: Valid Room Key:
Expected Outcome: The function should output a True statement in order to indicate a valid room key that exists in the larger file

5. Invalid Departure Date:
Input Parameters: Valid room key with an invalid departure date.
Expected Outcome: The function should raise a HotelManagementException with an error message indicating that the departure date is not registered or valid.

6. File Not Found (Checkout Data):
Input Parameters: N/A
Simulation: Used mock.patch to simulate the file not found scenario.
Expected Outcome: The function should raise a HotelManagementException with an appropriate error message indicating that the checkout data file is not found.

7. Integration Test:
Input Parameters: Valid room key.
Expected Outcome: The function should integrate correctly with other methods, such as validate_room_key and validate_departure_date, and successfully update the checkout data file.