GE-2 Part 2: Description of Test Cases

Function 1: Unit Testing
For testing function 1 we needed to test that room reservation was functional for each of the inputs into the function to make sure they were valid according to the assignment's standards. These inputs are Credit card number, ID card, Name and Surname, Phone Number, Room Type, Arrival Date, and the number of days. The equivalence classes and boundary values for each input are as follows:

Credit Card:
Valid Equivalence Classes: Follows Luhn's Algorithm correctness, data type is integer, integer is 16 digits
Invalid Equivalence Classes: Digit does not follow Luhn's Algorithm, more than 16 digits, less than 16 digits, and not a integer value
Invalid Boundary Values: 15 digit number and 16 digit number

ID Card:
Valid Equivalence Classes: Correctly follows Spanish DNI algorithm, 9 characters which are 8 integers followed by a letter, data type is string
Invalid Equivalence Classes: Does not follow Spanish DNI algorithm, more than 9 characters, less than 9 characters, not 8 characters followed by a letter, and data type is not a string
Invalid Boundary Values: 8 character input and 10 character input

Name and Surname:
Valid Equivalence Classes: String in between 10 and 50 characters, two strings with a space
Invalid Equivalence Classes: Not of datatype string, string with no spaces, over 50 characters, under 50 characters
Valid Boundary Values: 10 character string, 11 character string, 49 character string, 50 character string
Invalid Boundary Values: 9 character string, 51 character string

Phone Number:
Valid Equivalence Classes: Integer datatype, 9 digits
Invalid Equivalence Classes: Not an integer datatype, more than 9 digits, less than 9 digits
Invalid Boundary Values:

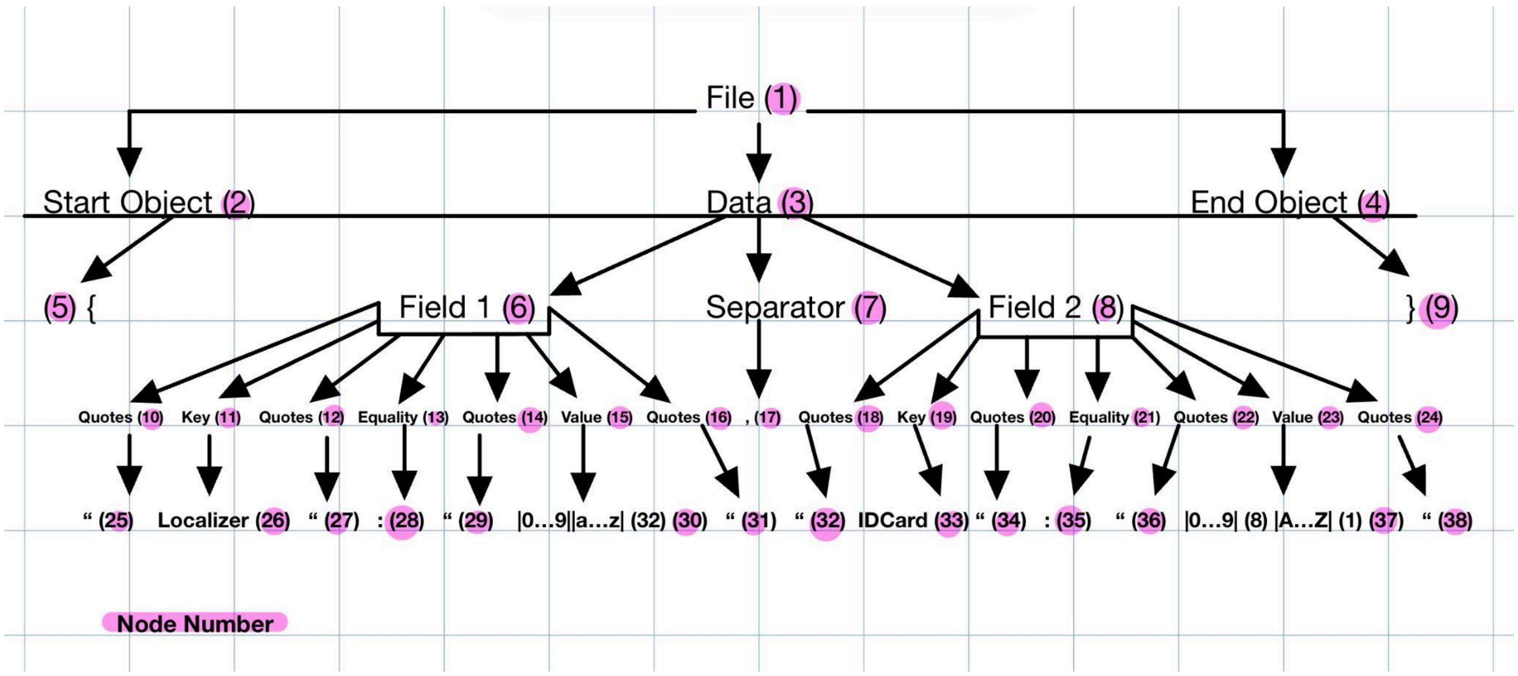
Room Type:
Valid Equivalence Classes: String, value "single", value "double", value "suite"
Invalid Equivalence Classes: Not a string of the above valid strings

Arrival Date:
Valid Equivalence Classes: String, in "DD/MM/YYYY" format, on/after current date
Invalid Equivalence Classes: Not a string, incorrect format, before current date, date does not exist on calendar

Num Days
Valid Equivalence Classes: Integer value, integer between 1 and 10

Invalid Equivalence Classes: Not and integer, integer below 1, integer above 10
Valid Boundary Values: 1, 2, 9, 10
Invalid Boundary Values: 0, 11

Function 2: Syntax Analysis



Grammar and Derivation Tree

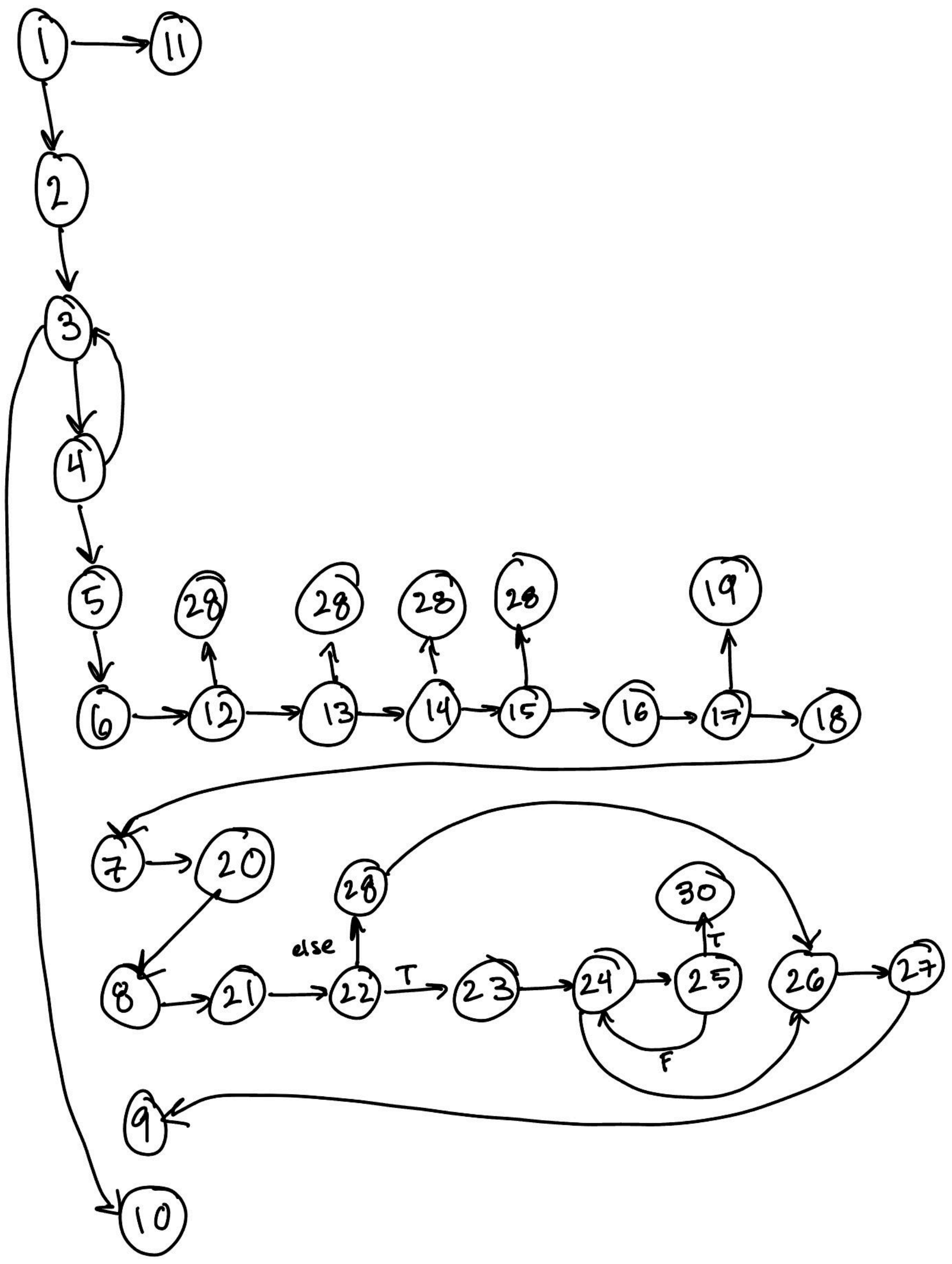
This tree divides up each individual component of the JSON file, allowing us to isolate each and test validity through deletion, duplication, and modification of each node. The test cases consist of taking a valid base case, and then testing the result of individually duplicating and deleting each node. If an identical test case for a node already exists for a higher node, we can exclude that case, and additionally, we create a test case for the modification of terminal nodes. The result of these tests depend on which nodes are altered. The valid case should return a valid SHA-256 string, altering either of the key or value nodes should still result in valid JSON syntax and return in invalid JSON key error or a localizer/idcard not found error, respectively. Altering any other nodes should invalidate the JSON syntax, and return a JSON decoding error.

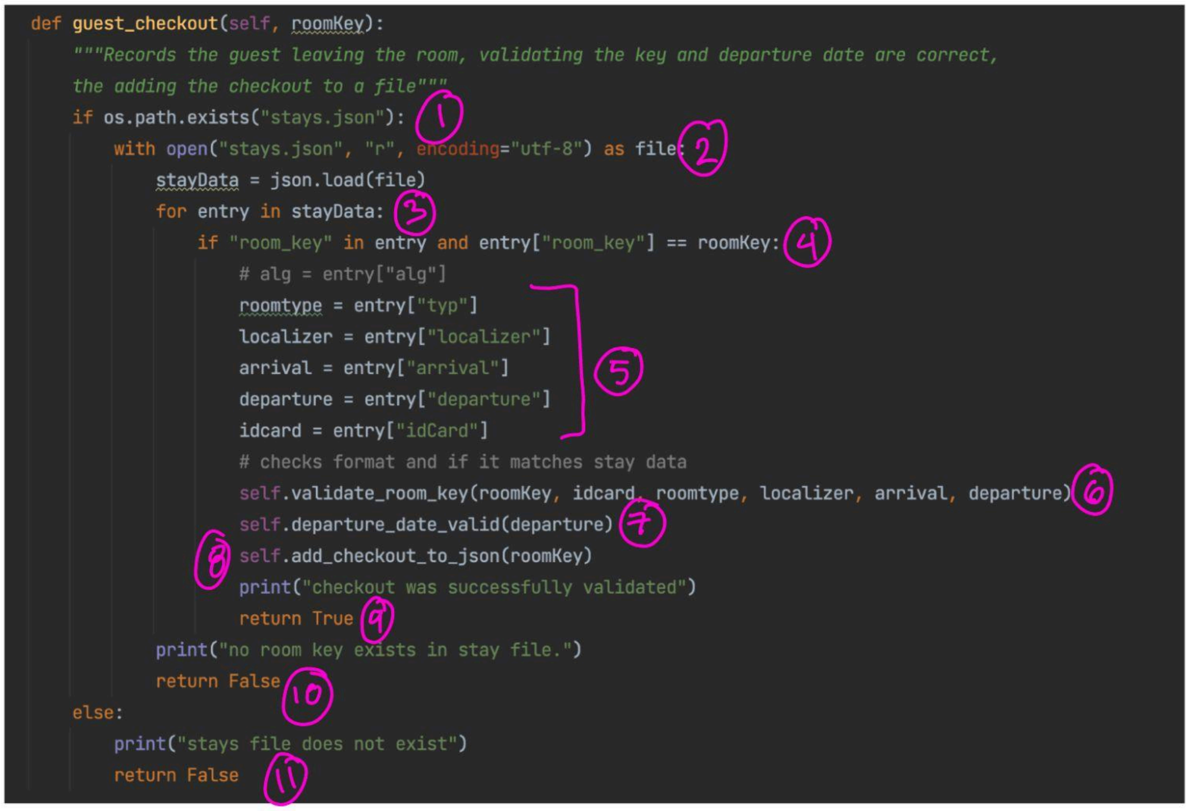
Additionally we need to test that the file has not been manipulated and that all of the data in the file corresponds to the localizer it outputs.

Function 3: Structural Testing

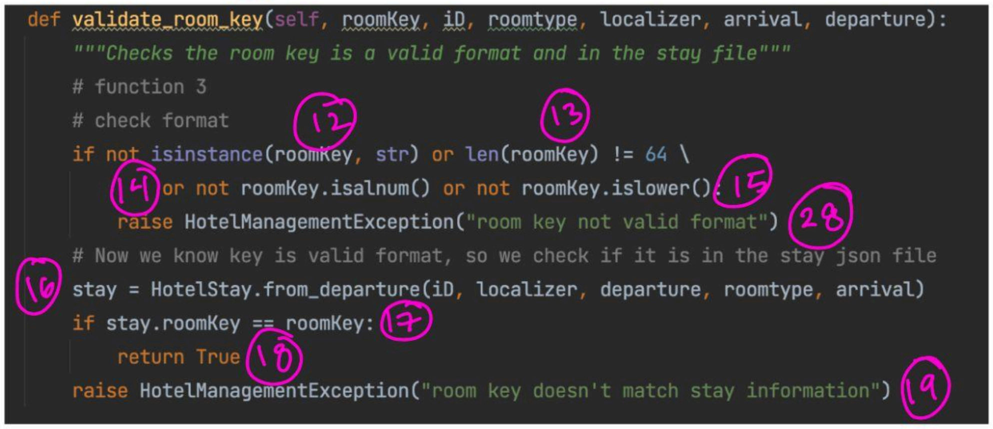
For structural testing we created a control flow tree to test all of the possible paths for the guest checkout function. Within the guest checkout function we had three helper functions each of which had their own tree which we combined with the larger tree, these functions being validate room key, validate departure date, and add checkout to json. We analyzed every possible path that an input could take along this function based on the input given.

This function had one for loop in the main function and one for loop in the add check out to json helper function. For testing the loop at node 3 in the main function, we needed to include test cases of the loop failing immediately due to no entries, the loop going back to node 3 due to the room key not matching the given room key, and the loop executing completely. We also had to test a look in our add checkout to json function where we had to test the loop leaving on the first time or going back through. Below, on the next page you can find the control flow graph for our functions, with the node numbering.

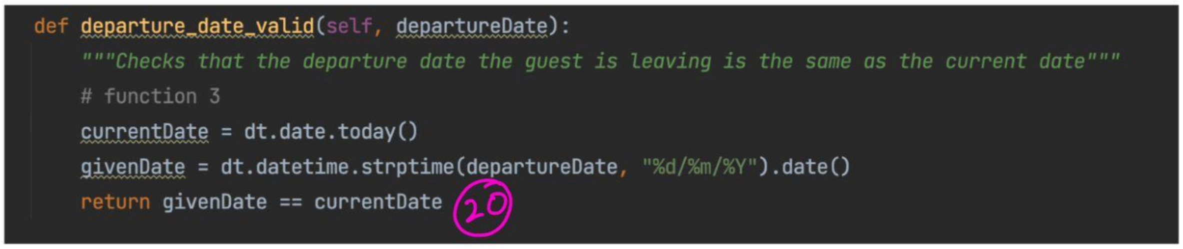

1
11
2
3
4
5
6
12
13
14
15
16
17
18
28
28
28
28
19
7
20
8
21
22
23
24
25
26
27
28
30
else
T
T
F
9
10



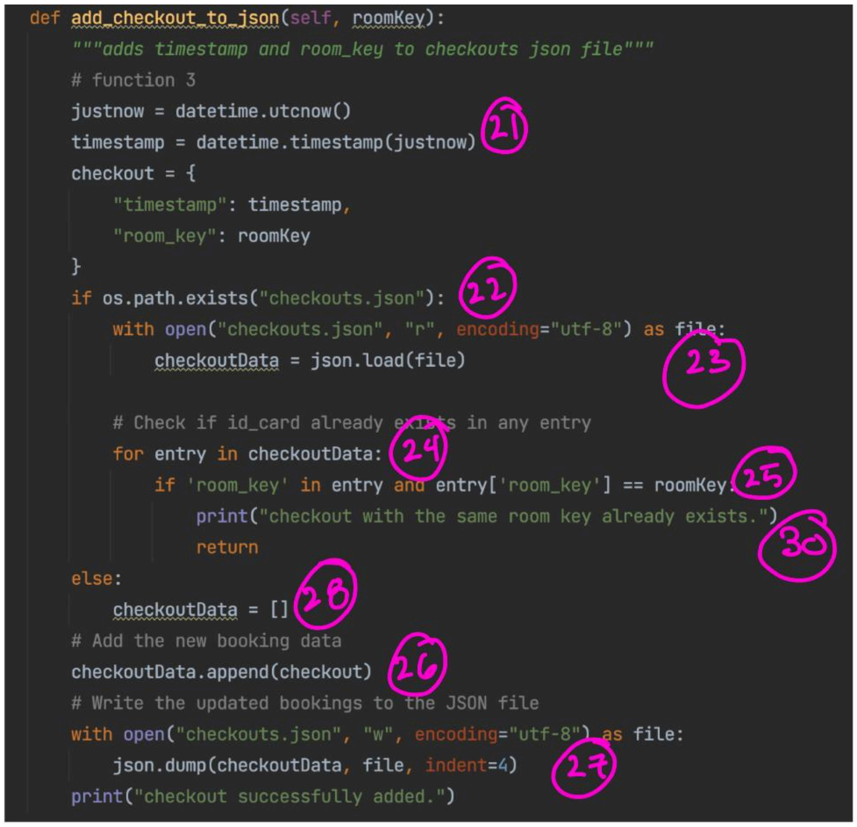
```
def guest_checkout(self, roomKey):
    """Records the guest leaving the room, validating the key and departure date are correct,
    the adding the checkout to a file"""
    if os.path.exists("stays.json"):   1
        with open("stays.json", "r", encoding="utf-8") as file:   2
            stayData = json.load(file)
            for entry in stayData:   3
                if "room_key" in entry and entry["room_key"] == roomKey:   4
                    # alg = entry["alg"]
                    roomtype = entry["typ"]
                    localizer = entry["localizer"]
                    arrival = entry["arrival"]            5
                    departure = entry["departure"]
                    idcard = entry["idCard"]
                    # checks format and if it matches stay data
                    self.validate_room_key(roomKey, idcard, roomtype, localizer, arrival, departure)   6
                    self.departure_date_valid(departure)   7
             8      self.add_checkout_to_json(roomKey)
                    print("checkout was successfully validated")
                    return True   9
            print("no room key exists in stay file.")
            return False   10
    else:
        print("stays file does not exist")
        return False   11
```



```
def validate_room_key(self, roomKey, iD, roomtype, localizer, arrival, departure):
    """Checks the room key is a valid format and in the stay file"""
    # function 3
    # check format              12                 13
    if not isinstance(roomKey, str) or len(roomKey) != 64 \
        14 or not roomKey.isalnum() or not roomKey.islower():   15
        raise HotelManagementException("room key not valid format")   28
    # Now we know key is valid format, so we check if it is in the stay json file
16  stay = HotelStay.from_departure(iD, localizer, departure, roomtype, arrival)
    if stay.roomKey == roomKey:   17
        return True   18
    raise HotelManagementException("room key doesn't match stay information")   19
```



```
def departure_date_valid(self, departureDate):
    """Checks that the departure date the guest is leaving is the same as the current date"""
    # function 3
    currentDate = dt.date.today()
    givenDate = dt.datetime.strptime(departureDate, "%d/%m/%Y").date()
    return givenDate == currentDate   20
```



```
def add_checkout_to_json(self, roomKey):
    """adds timestamp and room_key to checkouts json file"""
    # function 3
    justnow = datetime.utcnow()
    timestamp = datetime.timestamp(justnow)  21
    checkout = {
        "timestamp": timestamp,
        "room_key": roomKey
    }
    if os.path.exists("checkouts.json"):  22
        with open("checkouts.json", "r", encoding="utf-8") as file:  23
            checkoutData = json.load(file)

        # Check if id_card already exists in any entry
        for entry in checkoutData:  24
            if 'room_key' in entry and entry['room_key'] == roomKey:  25
                print("checkout with the same room key already exists.")  30
                return
    else:
        checkoutData = []  28
    # Add the new booking data
    checkoutData.append(checkout)  26
    # Write the updated bookings to the JSON file
    with open("checkouts.json", "w", encoding="utf-8") as file:
        json.dump(checkoutData, file, indent=4)  27
    print("checkout successfully added.")
```