Función 1 - Solicitar una reserva de hotel

La realización de las reservas se hace a través de la función denominada "room_reservation" dentro de la clase hotel_manager.
La función necesita 7 argumentos:

1. Número de tarjeta de crédito
2. Número de DNI con la letra correspondienta
3. Nombre y apellido(s)
4. Número de teléfono
5. Tipo de habitación
6. Fecha de llegada al hotel
7. Número de días de estancia en el hotel.

En primer lugar, se comprueba que todos los datos introducidos son correctos y cumplen con las condiciones necesarias.

- Comprobación de la tarjeta de crédito → la comprobación principal se realiza mediante una función realizada dentro de la misma clase denominada "validatecreditcard" que compruba a través de la librería Luhn que se trata de una tarjeta válida. Asimismo, para poder crear excepciones distintas, se comprueba antes de llamar a la función de validación, que la tarjeta no cuenta con más de 16 dígitos ni menos ni contiene letras. Para cualquiera de estos casos, si entra en ellos lanza una excepción a través de la función hotel_management_exception.
- Comprobación del número de DNI → realiza la comprobación por medio de una función denominada "validateidcard" realizada dentro de la misma clase. Esta, comprueba que la longitud sea exactamente igual a 9, que esté formado por 8 pfrimeros númros y una última letra y que cumple con el criterio de cálculo de la letra. En caso de devolver False, la función room_reservation lanza un error a través de hotel_management_exception
- Comprobación del nombre y apellido → en primer lugar se comprueba que tenga al menos dos palabras, así como que tenga entre 10 y 50 caracteres, que no empece ni termine por espacio, que no contenga dos espacios seguidos y que no contenga números. En caso de detectar un error, lanza un mensaje de error correspondiente por medio de hotel_management_exception.
- Número de teléfono → Para este argumento se realizan 3 comprobaciones, que no tiene ni más ni menos de 9 dígitos y que únicamente contenga números. En caso contrario, lanza un raise hotel_management_exception.
- Tipo de habitación → comprueba que sea de tipo "single", "double" o "suite" únicamente, en caso contrario, lanza error mediante hotel_management_exception.
- Fecha de llegada al hotel → para la comprobación, se ha realizado una función a parte denominada "validatearrival" que haría saltar un mensaje de error en caso de problema. Esta función comprueba que la fecha sigue el formato DD/MM/AAAA, es decir, que está formada por 3 secciones, día, mes y año, teniendo dos dígitos, sin contener letras, los dos primeros y 4 el último, separados por "/". Además, comprueba los meses que tienen que tener 30 y 31 años están correctamente, al igual de 28 o 29 días para febrero en función de si es año bisiesto o no.
- Número de días en el hotel → lo hace a través de una función denominada "validatenumdays" que comprueba que se trata de un número entre 1 y 10, en caso contrario hace un raise de hotel_management_exception.

- No hay una reserva ya hecha → Para comprobar que una persona con ese DNI no ha realizado ya la reserva, abre el fichero de reservas (reservas.json), lo lee a través de read_data_from_jason y busca si para la clave “id_card” ya hay una con el DNI introducido como argumento, en caso afirmativo, lanza mensaje de error a través de hotel_management_exception.

Después de realizar todas estas comprobaciones, si continua en la función significa que se trata de un caso válido, por lo que creamos un localizador mediante una llamda a hotel_reservation con los datos correspondientes.
Posteriormente se almacena la reserva en el fichero “reservas.json” poniendo cada clave con el argumento indicado, mediante un append y haciendo referencia a la función write_data_to_json.
Por último, devuelve el localizador.

Tests
El “archivo test_room_reservation_tests” contine la clase “test_room_reservation” que permite realizar los tests para la función “room_reservation”.
Tras definir el path donde se encuentra el archivo tests1.json que contiene los casos de prueba, comienza con la función set up.
Para la función “setUp”, se abre el archivo json tests1.json y se cargan los datos, salvo no encontrarlo que hace un raise del hotel_management_exception, o que no exista, que lo crea vacío.
Para la realización de los tests, se han creado dos funciones, una para los casos válidos y otra para incorrectos.

- Función para casos válidos “test_reservatio_ok” → para los casos a los que llama el set up, seleciona los tests 1, 6, 8, 13, 17, 18, 19, 21 y 31 y para cada uno llama a la clase hotel_manager y guarda en la variable “localizer” el lozalizador que devuelve al ejecutar la función “room_reservation”.
  Para cada uno de estos tests comprueba que el localizador es efectivamente el que corresponde para esos datos.
  Estos casos válidos verifican:
  - Caso 1 → Caso de equivalencia 1 con tarjeta de crédito que cumple el algoritmo de Luhn, caso de equivalencia 2 con el tipo de datos correctos y valor límite válido 1 con una tarjeta de 16 dígitos exactos.
  - Caso 6 → comprueba que el DNI cumple con todos los criterios necesarios comprobados en la función “validateidcard” del hotel_manager ya mencionada.
  - Caso 8 → Caso de equivalencia válido 5, se trata de tipo de datos para nombre y apellidos correcto, caso de equivalencia válido 6, verificando que hay al menos un espacio y valor límite válido 2, que cumple una longitud entre 10 y 50 exactamente.
  - Caso 13 → Valor límite válido 3 que comprueba la longitud del número de teléfono es exáctamente 9, y caso de equivalencia válido 7 que comprueba que son todos números.
  - Caso 17 → caso de equivalencia válido 8, verifica que el tipo de habitación es “single”.
  - Caso 18 → caso de equivalencia válido 8, verifica que el tipo de habitación es “double”.

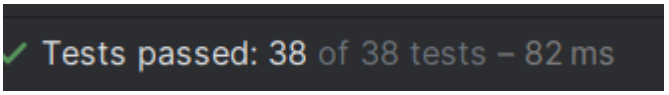
  - Caso 19 → caso de equivalencia válido 8, verifica que el tipo de habitación es “suite”.
  - Caso 21 → caso de equivalencia válido 9, que verifica que la fecha de llegada cumple con el formato especificado en la funcion “validatearrival”.
  - Caso 31 → caso de equivalencia válido 10 que verifica que el número de días cumple con el formato de la función “validatenumdays”.

  Tras verificar todos estos casos, lo tranforma a True y lo comprueba.

- Función con los casos no válidos “test_reservation_ko” → Comprueba los tests restantes de igual manera que en los casos válidos, para los casos restantes.
  Los casos son los siguientes:
  - Caso 2 → Caso de equivalencia no válido 1, con una tarjeta de 16 números pero que no cumple el algoritmo de Luhn
  - Caso 3 → Caso de equivalencia no válido 2, con una tarjeta de crédito que contiene una letra
  - Caso 4 → Caso de equivalencia no válido 3, con una tarjeta de crédito con 17 números, y valor límite no válido 1
  - Caso 5 → Caso de equivalencia no válido 4, tarjeta de crédito de 15 números, y valor límite no válido 1
  - Caso 7 → Caso de equivalencia no válido 5, con un dni que no cumple el criterio de asignación de la letra en función de los números
  - Caso 9 → Caso de equivalencia no válido 8, nombre y apellido de 7 dígitos y valor límite no válido 8
  - Caso 10 → Caso de equivalencia no válido 6, con un apellido que contiene números
  - Caso 11 → Caso de equivalencia no válido 7, con un nombre y apellido de más de 50 dígitos y valor límite no válido 7
  - Caso 12 → Caso de equivalencia no válido 9, caso de nombre y apellido sin espacios
  - Caso 14 → Caso de equivalencia no válido 12, con número de teléfono que contiene letras
  - Caso 15 → Caso de equivalencia no válido 11, con número de teléfono de más de 9 números, y valor límite no válido 9
  - Caso 16 → Caso de equivalencia no válido 10, con número de teléfono de menos de 9 números, y valor límite no válido 10
  - Caso 20 → Caso de equivalencia no válido 13, con tipo de habitación “triple” inválida
  - Caso 22 → Caso de equivalencia no válido 14, con fecha de llegada con letras
  - Caso 23 → Caso de equivalencia no válido 15, con año de fecha de llegada de más de 4 dígitos, y valor límite no válido 11
  - Caso 24 → Caso de equivalencia no válido 16, con mes de fehca de llegada superior a 12
  - Caso 25 → Caso de equivalencia no válido 17, con dia de fecha de llegada mayor a 31
  - Caso 26 → Caso de equivalencia no válido 18, con fecha de llegada no separada por “/”

- Caso 27 → Caso de equivalencia no válido 19, con fecha de llegada con un día de más de 1 dígito, y valor límite no válido 15
- Caso 28 → Caso de equivalencia no válido 19, con fecha de llegada con un día de 3 dígitos, y valor límite no válido 14
- Caso 29 → Caso de equivalencia no válido 20, con fecha de llegada con un mes de 1 dígito, y valor límite no válido 13
- Caso 30 → Caso de equivalencia no válido 20, con fecha de llegada con un mes de 3 dígitos, y valor límite no válido 12
- Caso 32 → Caso de equivalencia no válido 21, con número de días de estancia de 0.5, y valor límite no válido 16
- Caso 33 → Caso de equivalencia no válido 22, con número de días de estancia de 45, y valor límite no válido 17
- Caso 34 → Caso de equivalencia no válido 23, con número de días con letras
- Caso 35 → Caso de equivalencia no válido 24 porque contiene dos espacios seguidos entre el nombre y el apellido
- Caso 36 → Caso de equivalencia no válido 25 donde comienza por espacio

Dan correctos todos los casos:

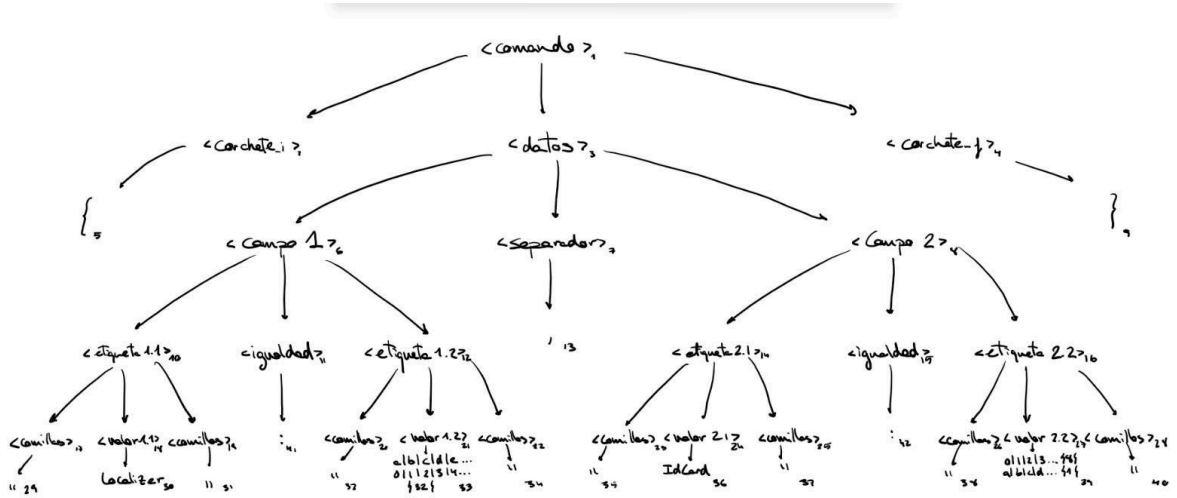


Función 2 - llegada al hotel
Una vez llegado al hotel, la función “guest_arrival”, recibe el DNI y la tarjeta de crédito y genera un código de habitación.
Para la realización de esta función, se carga el fichero introducido y salta un error en caso de estar vacío.
A partir de ahí, realiza las siguientes comprobaciones:

- Comprueba que las claves sean “localizer” e “IdCard”
- Si la información de “localizer” o de “IdCard” está vacía
- Si el localizador no tiene longitud de 32 o el dni no tiene longitud de 9
- Si el formato del localizar o del DNI no es el correcto
- Si el json no tiene el formato adecuado

A continuación, lee el archivo “reservas” generado por la función 1, busca la clave “localizador” y busca el introducido, si no lo encuentra lanza una excepción de error. Una vez lo encuentra, comprueba que el dni es el correcto, enb caso contrario hace un raise de hotel_management_exception.
Una vez encontrados los datos, calcula la fecha de salida “departure”, almacena los datos y crea una room_key que almacena en el fichero json “estancias”.

Tests
Para la realización de los tests, nos basamos en el árbol de derivación siguiente:



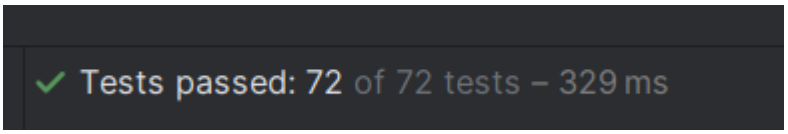
Al cual le corresponde la gramática siguiente:
<comando> ::= <corchete_i> <datos> <corchete_f>
<corchete_i> ::= {
<corchete_j> ::= }
<datos> ::= <campo1> <separador> <campo2>
<campo1> ::= <etiqueta1.1> <igualdad> <etiqueta1.2>
<separador> ::= ,
<campo2> ::= <etiqueta2.1> <igualdad> <etiqueta2.1>
<etiqueta1.1> ::= <comillas> <valor1.1> <comillas>
<igualdad> ::= :
<etiqueta1.2> ::= <comillas> <valor1.2> <comillas>
<etiqueta2.1> ::= <comillas> <valor2.1> <comillas>
<etiqueta2.1> ::= <comillas> <valor2.2> <comillas>
<comillas> ::= “
<valor 1.1> ::= localizer
<valor 1.2> ::= a|b|c|d|e|...|0|1|2|3… {32}
<valor 2.1> ::= IdCard
<valor 1.2> ::= 0-9 {8} a-z{1}

Así, se han diseñado 70 tests para cada uno de los casos presentes en el esquema, todos ellos contemplados en el excel.
El fichero de comprobaciones para la función 2 se denomina “test_guest_arrival”. Comienza, al igual que lo tests de la función anterior, almacenando el path de dónde se encuentran los ficheros json con las pruebas.
Continua con la función set up que busca el fichero “estancias”, donde la función 2 almacena sus resultados, y lo borra en caso de existir.
Le sigue la función “get_store_hash” que abre el fichero reservas salvo excepción.
Para las comprobaciones de los tests se realiza a través de dos funciones, una para los casos válidos y otra para los inválidos.

- Función de casos válidos, “test_reservation_ok” → abre el test válido 1, obtiene la información del archivo de pruebas, llama a la función “guest:arrival” y compara la roomkey que se obtiene con la que debería.

- Función de casos no válidos, “test_reservation_ko” → al igual que ocurre en la función descrita anteriormente, los errores contemplados en los casos de prueba son los siguientes:
    - Fallos de escritura de alguna de las claves
    - Longitud de alguna de las etiquetas incorrecta
    - Formato erróneo en alguna de las etiquetas
    - Fichero “reservas” vacío
    - Valor de alguna de las 2 etiquetas es nulo
    - Formato de JSON incorrecto

Funcionan todos los tests:

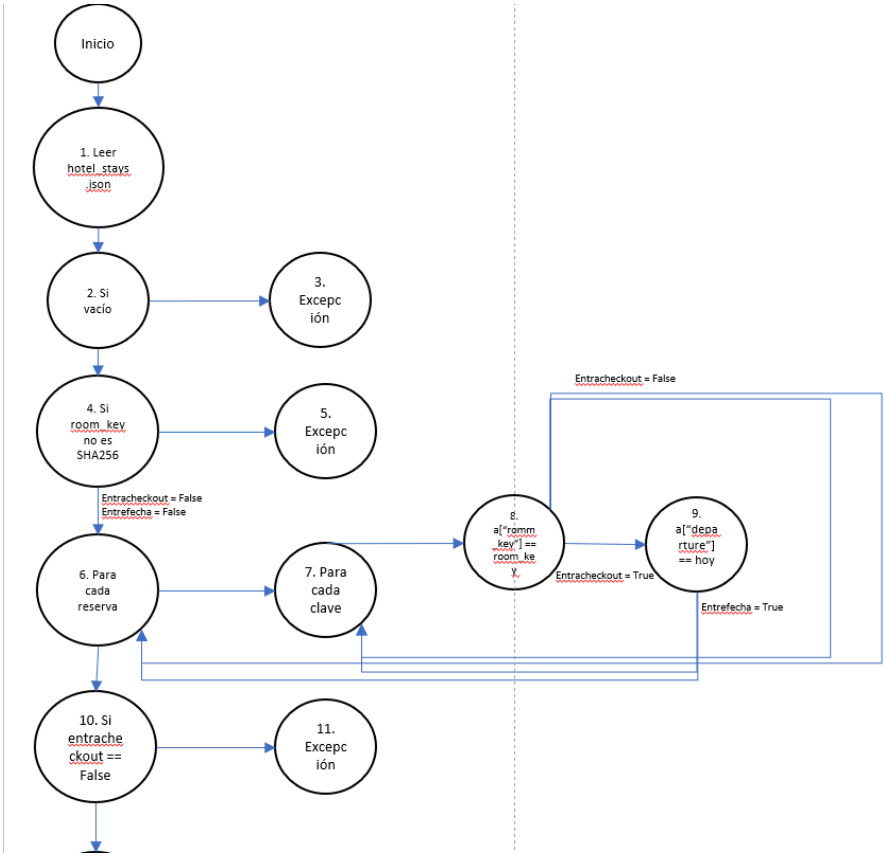


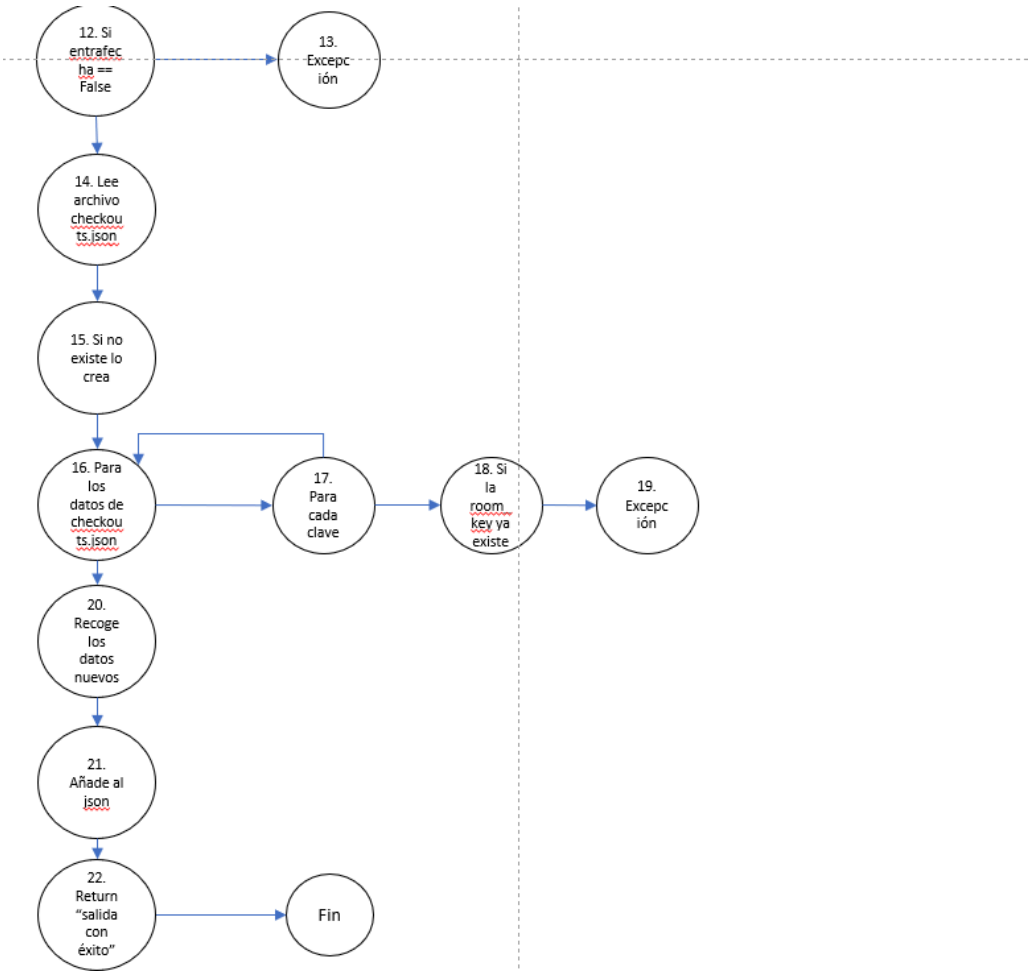
Función 3

La función guest_departure se encarga de gestionar el proceso de salida de un huésped del hotel. Lo hace de la siguiente manera:

- Verificación de datos de estancia → En primer lugar, la función comprueba si existen datos de estancias disponibles. Esto se hace mediante la lectura de un archivo JSON que contiene información sobre las estancias de los huéspedes en el hotel. Si el archivo está vacío o no existe, se levanta una excepción de tipo hotel_management_exception.
- Verificación del formato de la room_key→ Después de confirmar la existencia de datos de estancias, la función procede a verificar si la room_key proporcionada cumple con un formato específico. Utiliza una expresión regular para esto, buscando una cadena hexadecimal de 64 caracteres de longitud. Si la room_key no cumple con este formato, se levanta una excepción.
- Búsqueda de la room_key en los datos de estancia→ Una vez que se ha verificado el formato de la room_key, la función busca esta clave en los datos de estancias almacenados en el archivo JSON. Si la room_key no se encuentra en los datos de estancia, se levanta una excepción.
- Verificación de la fecha de salida→ Si se encuentra la room_key en los datos de estancia, la función verifica si la fecha de salida esperada coincide con la fecha actual. Primero, extrae la fecha actual en formato de cadena y la compara con la fecha de salida esperada para la estancia correspondiente. Si la fecha de salida no coincide con la fecha actual, se levanta una excepción.
- Almacenamiento de la salida del huésped→ Si la room_key existe y la fecha de salida coincide con la fecha actual, la función procede a almacenar los datos de salida del huésped. Primero, lee los datos almacenados en un archivo JSON que registra las salidas de los huéspedes. Si no hay datos disponibles, se inicializa una lista vacía. Luego, verifica si el huésped ya ha realizado el proceso de salida en el mismo día. Si es así, se levanta una excepción. En caso contrario, se añaden los datos de salida del huésped a la lista y se escriben en el archivo JSON.
- Una vez completadas todas las verificaciones y acciones necesarias, la función retorna True para indicar que la salida del huésped se ha procesado correctamente.

A partir de esta función realizamos el siguiente grafo para la realización de los tests siguiendo el esquema de la función anterior:





Así sacamos los tests para probar:

- Caso correcto
- room_key no cumple el formato correcto
- room_key no está entre los datos de estancias
- Fecha de salida diferente a la actual
- room_key ya está en el archivo checkouts

Problemas encontrados:
Tenemos un error común en los tests 2, 3, 4, 5, 6 que si que entra en el mensaje correcto, sin embargo no sale del raise hotel_management_exception, como se puede apreciar en el siguiente error:

```
Ejecutando: TC6
id_test TC6
la fecha de llegada es:  2024-06-16 00:00:00
la fecha de salida es: 18/06/2024
SubTest error: Traceback (most recent call last):
  File "C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2288.0_x64__qbz5n2kfra8p0\Lib\unittest\case.py"
    yield
  File "C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2288.0_x64__qbz5n2kfra8p0\Lib\unittest\case.py"
    yield
  File "C:\Users\ghija\PycharmProjects\6801.2024.grupo.2.EG2\src\unittest\python\test_check_out.py", line 61, in test_check_ou
    hm.guest_departure(inputData["room_key"])
  File "C:\Users\ghija\PycharmProjects\6801.2024.grupo.2.EG2\src\main\python\uc3m_travel\hotel_manager.py", line 362, in guest
    raise hme("La persona ya ha hecho checkout hoy")
hotel_management_exception.hotel_management_exception: La persona ya ha hecho checkout hoy
```