In [1]:
```
#MAESTRÍA EN INTELIGENCIA ARTIFICIAL APLICADA
#Pruebas de software y aseguramiento de la calidad
#Dr. Gerardo Padilla Zárate

#Actividad 6.2 Ejercicio de programación 3 y pruebas de unidad
#CARLOS ENRIQUEZ GORGONIO
#A01793102
#21 de febrero de 2024
```

In [2]:
```
'''Actividad 1. Reservation System
Descripción
Req 1. Implement a set of classes in Python that implements two abstractions:
    1. Hotel
    2. Reservation
    3. Customers
Req 2. Implement a set of methods to handle the next persistent behaviors (stored in files):
    1. Hotels
        a. Create Hotel
        b. Delete Hotel
        c. Display Hotel information
        d. Modify Hotel Information
        e. Reserve a Room
        f. Cancel a Reservation
    2. Customer
        a. Create Customer
        b. Delete a Customer
        c. Display Customer Information
        d. Modify Customer Information
    3. Reservation
        a. Create a Reservation (Customer, Hotel)
        b. Cancel a Reservation You are free to decide the attributes within each  class that enable the required behavior.
Req 3. Implement unit test cases to exercise the  methods in each class. Use the unittest module in Python.
Req 4. The code coverage for all unittests should accumulate at least 85% of line coverage.
Req 5. The program shall include the mechanism to handle invalid data in the file. Errors should be displayed in the console and the execution must continue.
Req 6. Be compliant with PEP8.
Req 7. The source code must show no warnings using Fleak and PyLint.'''
```

Out[2]:
```
'Actividad 1. Reservation System\nDescripción\nReq 1. Implement a set of classes in Python that implements two abstractions:\n    1. Hotel\n    2. Reservation\n    3. Customers\nReq 2. Implement a set of methods to handle the next persistent behaviors (stored in files):\n    1. Hotels\n        a. Create Hotel\n        b. Delete Hotel\n        c. Display Hotel information\n        d. Modify Hotel Information\n        e. Reserve a Room\n        f. Cancel a Reservation\n    2. Customer\n        a. Create Customer\n        b. Delete a Customer\n        c. Display Customer Information\n        d. Modify Customer Information\n    3. Reservation\n        a. Create a Reservation (Customer, Hotel)\n        b. Cancel a Reservation You are free to decide the attributes within each  class that enable the required behavior.\nReq 3. Implement unit test cases to exercise the  methods in each class. Use the unittest module in Python.\nReq 4. The code coverage for all unittests should accumulate at least 85% of line coverage.\nReq 5. The program shall include the mechanism to handle invalid data in the file. Errors should be displayed in the console and the execution must continue.\nReq 6. Be compliant with PEP8.\nReq 7. The source code must show no warnings using Fleak and PyLint.'
```

In [3]:
```
#!pip install pylint
#!pip install pylint[spelling]
#!pip install flake8
```

In [4]:
```
import json
import os
import unittest
```

In [5]:
```
#Primer sección - Clases de errores
```

In [6]:
```
#Definimos el error en caso de archivos corruptos
class error(Exception):
    def __init__(self, ruta):
        alerta = (f"El archivo esta corrupto:{ruta}")
        super().__init__(alerta)
        self.alerta = alerta
```

In [7]:
```python
#Definimos el error en caso de no encontrar un hotel por su ID
class error_hotel(Exception):
    def __init__(self, numero_hotel):
        alerta = f"No se encuentra el hotel numero{numero_hotel}"
        super().__init__(alerta) #(message)
        self.alerta = alerta     #señf.message
```

In [8]:
```python
#Definimos el error en caso de no encontrar un cliente buscado
class error_huesped(Exception):

    def __init__(self, numero_huesped):
        alerta = f"No se encuentra el usuario con el ID proporcionado{numero_huesped}"
        super().__init__(alerta)
        self.alerta = alerta
```

In [9]:
```python
#Definimos el error de no encontrar un hotel para hacer una reserva en este
class error_hotel_r(Exception):

    def __init__(self, numero_hotel):
        alerta = (f"Numero de hotel invalido para reservaciones:{numero_hotel}")
        super().__init__(alerta)
        self.alerta = alerta
```

In [10]:
```python
#Definimos el error de no encontrar un usuarios para hacer una reserva en este
class error_huesped_r(Exception):

    def __init__(self, numero_huesped):
        alerta = (f"Numero de huesped invalido para reservas {numero_huesped}")
        super().__init__(alerta)
        self.alerta = alerta
```

In [11]:
```python
#Definimos el error de no encontrar un numero de reserva
class error_reserva(Exception):

    def __init__(self, numero_reserva):
        alerta = f"Reservation not found for ID {numero_reserva}"
        super().__init__(alerta)
        self.alerta = alerta
```

In [12]:
```python
#Segunda sección - Funciones para hoteles incisos a-f
```

In [13]:

```python
#Elemento Hotel -- acción crear nuevo registro o añadir
#Creamos la función que nos permite crear un nuevo hotel
def nuevo_hotel(nombre_nuevo_hotel):

    #Definimos el nombre que le daremos al archivo
    archivo_hoteles = "hoteles.json"
    #Inicializamos los numeros de los hoteles registrados, sirve como ID
    indice = 0

    try:
        #declaramos el contenedor de nuestra lista de hoteles
        lista_hoteles = []
        #Si no hay archivo lo creamos, de lo contrario lo abrimos, ordenamos y añadimos valores
        if not os.path.exists(archivo_hoteles):
            with open(archivo_hoteles, "w", encoding="utf-8") as file:
                print(json.dumps([]), file=file)
        else:
            #Si el archivo existe, se abre en modo lectura
            with open(
                archivo_hoteles, 'r', encoding="utf-8") as file:
                archivo_lista = json.load(file)
                if len(archivo_lista) > 0:
                    lista_hoteles = sorted(archivo_lista,key=lambda x: x['ID_HOTEL'])
                    indice = lista_hoteles[len(lista_hoteles) - 1]['ID_HOTEL']

        #Añadimos el hotel a nuestra lista
        hotel = {}
        hotel['ID_HOTEL'] = indice + 1
        hotel['NOMBRE_HOTEL'] = nombre_nuevo_hotel
        lista_hoteles.append(hotel)

        with open(archivo_hoteles, "w", encoding="utf-8") as file:
            print(json.dumps(lista_hoteles), file=file)

        return hotel

    #En caso de errores, llamamos a nuestra función predefinida
    except json.JSONDecodeError as e:
        raise error(archivo_hoteles) from e
```

In [14]:
```python
#Elemento Hotel -- borrar un registro de hotel, con base en su numero de indice
#Creamos la función que nos permite borrar un hotel de nuestro archivo
def borrar_hotel(indice_hotel):

    #ubicamos nuestro archivo general con registro de datos de hoteles
    archivo_hoteles = "hoteles.json"

    try:
        #Abrimos el archivo y buscamos el ID introducido
        with open(archivo_hoteles, 'r', encoding="utf-8") as archivo:
            lista_cargada = json.load(archivo)
            if len(lista_cargada) == 0:
                raise error_hotel(indice_hotel)

            diccionario_abierto = dict(
                (li['ID_HOTEL'], li) for li in lista_cargada
            )
            if indice_hotel not in diccionario_abierto:
                raise error_hotel(indice_hotel)

            lista_cargada.clear()
            #Despues de encontrar el archivo en la lista, lo eliminamos
            del diccionario_abierto[indice_hotel]
            lista_cargada.extend(list(diccionario_abierto.values()))

            with open(archivo_hoteles, "w", encoding="utf-8") as file:
                print(json.dumps(lista_cargada), file=file)

    except FileNotFoundError as exc:
        raise error_hotel(indice_hotel) from exc

    except json.JSONDecodeError as e:
        raise error(archivo_hoteles) from e
```

In [15]:
```python
#Elemento Hotel -- acción mostrar información de un hotel con base en su indice o ID
#Creamos la función que nos permite mostrar un hotel con base en su indice
def mostrar_hotel(indice_hotel):

    archivo_hoteles = "hoteles.json"
    try:
        with open(
            #Abrimos el archivo en modo lectura para buscar elementos
            archivo_hoteles, 'r', encoding="utf-8"
        ) as archivo:
            lista_cargada = json.load(archivo)
            if len(lista_cargada) == 0:
                raise error_hotel(indice_hotel)

            #Bucamos el hotel en lusta lista  cargada
            diccionario_abierto = dict(
                (hotel['ID_HOTEL'], hotel) for hotel in lista_cargada
            )
            if indice_hotel not in diccionario_abierto:
                raise error_hotel(indice_hotel)

            return diccionario_abierto.get(indice_hotel)

    except FileNotFoundError as exc:
        raise error_hotel(indice_hotel) from exc
    except json.JSONDecodeError as e:
        raise error(archivo_hoteles) from e
```

In [16]:
```python
#Elemento Hotel -- acción modificar información de un hotel con base en su indice o ID
#Creamos la función que nos permite crear un nuevo hotel
def modificar_hotel(indice_hotel, nuevo_nombre):

    archivo_hoteles = "hoteles.json"
    try:
        with open(archivo_hoteles, 'r', encoding="utf-8") as file:
            carga_elementos = json.load(file)
            if len(archivo_hoteles) == 0:
                raise error_hotel(indice_hotel)

            diccionario_busqueda = dict(
                (hotel['ID_HOTEL'], hotel) for hotel in carga_elementos
            )
            if indice_hotel not in diccionario_busqueda:
                raise error_hotel(indice_hotel)

            carga_elementos.clear()
            lista_modificada = {}
            lista_modificada['ID_HOTEL'] = indice_hotel
            lista_modificada['NOMBRE_HOTEL'] = nuevo_nombre
            diccionario_busqueda[indice_hotel] = lista_modificada
            carga_elementos.extend(list(diccionario_busqueda.values()))
            with open(archivo_hoteles, "w", encoding="utf-8") as file:
                print(json.dumps(carga_elementos), file=file)

            return lista_modificada

    except FileNotFoundError as exc:
        raise error_hotel(indice_hotel) from exc
    except json.JSONDecodeError as e:
        raise error(archivo_hoteles) from e
```

In [17]:
```python
#Tercera Sección - Funciones para huespedes (customers) incisos a-d
```

In [18]:

```python
#Elemento Huespedes -- acción crear nuevo huesped
#Definimos una función para crear nuevos usuarios
def nuevo_huesped(nombre_huesped):

    archivo_huespedes = "huespedes.json"
    #Creamos nuestra variable contenedora incial para el indice de huespedes
    indice_base = 0

    try:
        #Creamos nuestro contenedor de huespedes antes de abrir el archivo
        lista_huespedes = []
        if not os.path.exists(archivo_huespedes):
            with open(archivo_huespedes, "w", encoding="utf-8") as file:
                print(json.dumps([]), file=file)
        else:
            with open(archivo_huespedes, 'r', encoding="utf-8") as file:
                #Cargamos nuestra lista de indices o numeros de huespedes
                lista_cargada = json.load(file)
                if len(lista_cargada) > 0:
                    lista_huespedes = sorted(
                        lista_cargada,
                        key=lambda x: x['ID_HUESPED']
                    )
                    #Aumentamos nuestro numero de indice
                    indice_base = lista_huespedes[
                        len(lista_huespedes) - 1
                    ]['ID_HUESPED']

        huesped = {}
        huesped['ID_HUESPED'] = indice_base + 1
        huesped['NOMBRE_HUESPED'] = nombre_huesped
        lista_huespedes.append(huesped)

        with open(archivo_huespedes, "w", encoding="utf-8") as file:
            print(json.dumps(lista_huespedes), file=file)

        return huesped


    except json.JSONDecodeError as e:
        raise error(archivo_huespedes) from e
```

In [19]:
```python
#Elemento Huespedes -- acción eliminar huesped
#Definimos una función para eliminar un huesped con base en su numero de indice
def borrar_huesped(indice_huesped):

    archivo_huespedes = "huespedes.json"

    try:
        with open(
            archivo_huespedes, 'r', encoding="utf-8"
        ) as file:
            lista_cargada = json.load(file)
            if len(lista_cargada) == 0:
                raise error_huesped(indice_huesped)

            huespedes = dict(
                (huesped['ID_HUESPED'], huesped) for huesped in lista_cargada
            )
            if indice_huesped not in huespedes:
                raise error_huesped(indice_huesped)

            lista_cargada.clear()
            del huespedes[indice_huesped]
            lista_cargada.extend(list(huespedes.values()))
            with open(
                archivo_huespedes, "w", encoding="utf-8"
            ) as file:
                print(json.dumps(lista_cargada), file=file)


    except FileNotFoundError as exc:
        raise error_huesped(indice_huesped) from exc
    except json.JSONDecodeError as e:
        raise error(archivo_huespedes) from e
```

In [20]:
```python
#Elemento Huespedes -- acción mostrar información de huesped
#Definimos una función para crear nuevos usuarios

def mostrar_huesped(indice_huesped):

    archivo_huespedes = "huespedes.json"

    try:
        #Hacemos una busqueda en el archivo de huespedes
        with open(
            archivo_huespedes, 'r', encoding="utf-8"
        ) as file:
            lista_archivo = json.load(file)
            if len(lista_archivo) == 0:
                raise error_huesped(indice_huesped)

            #cargamos la lista donde se realizarán las iteraciones de busqueda
            lista = dict(
                (huesped['ID_HUESPED'], huesped) for huesped in lista_archivo
            )
            if indice_huesped not in lista:
                raise error_huesped(indice_huesped)

            return lista.get(indice_huesped)

    except FileNotFoundError as exc:
        raise error_huesped(indice_huesped) from exc
    except json.JSONDecodeError as e:
        raise error(archivo_huespedes) from e
```

In [21]:
```python
#Elemento Huespedes -- acción modificar huesped
#Definimos una función para modificar el nombre de un huesped buscado por indice
def modificar_huesped(indice_huesped, nuevo_nombre_huesped):

    archivo_huespedes = "huespedes.json"

    try:
        #cargamos y verificamos el contenido de nuestro archivo con huespedes
        with open(
            archivo_huespedes, 'r', encoding="utf-8"
        ) as file:
            lista_origen = json.load(file)
            if len(lista_origen) == 0:
                raise error_huesped(indice_huesped)

            lista_real = dict(
                (huesped['ID_HUESPED'], huesped) for huesped in lista_origen
            )
            if indice_huesped not in lista_real:
                raise error_huesped(indice_huesped)

            lista_origen.clear()

            lista_nueva = {}
            lista_nueva['ID_HUESPED'] = indice_huesped
            lista_nueva['NOMBRE_HUESPED'] = nuevo_nombre_huesped
            lista_real[indice_huesped] = lista_nueva
            lista_origen.extend(list(lista_real.values()))

            with open(
                archivo_huespedes, "w", encoding="utf-8"
            ) as file:
                print(json.dumps(lista_origen), file=file)

            return lista_nueva

    except FileNotFoundError as exc:
        raise error_huesped(indice_huesped) from exc
    except json.JSONDecodeError as e:
        raise error(archivo_huespedes) from e
```

In [22]:
```python
#Cuarte sección - Funciones para reservaciones incisos a-b
```

In [23]:

```python
#Elemento Reservacciones -- acción crear reservación
#Definimos una función para crear una reservación y/o un nuevo archivo de reservaciones
def nueva_reserva(indice_hotel, indice_huesped, inicio_reserva, fin_reserva):

    archivo_reservas = "reservas.json"

    try:
        lista_hoteles = []
        if not os.path.exists(archivo_reservas):
            with open(
                archivo_reservas, "w", encoding="utf-8"
            ) as file:
                print(json.dumps([]), file=file)
        else:
            with open(
                archivo_reservas, 'r', encoding="utf-8"
            ) as file:
                reservas_existentes = json.load(file)
                if len(reservas_existentes) > 0:
                    lista_hoteles = sorted(
                        reservas_existentes,
                        key=lambda x: x['ID_HOTEL']
                    )
        try:
            mostrar_hotel(indice_hotel)
        except error_hotel as ex:
            raise error_hotel_r(indice_hotel) from ex
        try:
            mostrar_huesped(indice_huesped)
        except error_huesped as ex: #-------------------------------
            raise error_huesped_r(indice_huesped) from ex

        #Creamos nuestra reserva y la añadimos al archivo
        reserva = {}
        reserva['ID_HOTEL'] = indice_hotel
        reserva['ID_HUESPED'] = indice_huesped
        reserva['FECHA_ENTRADA'] = inicio_reserva
        reserva['FECHA_SALIDA'] = fin_reserva
        lista_hoteles.append(reserva)

        with open(
            archivo_reservas, "w", encoding="utf-8"
        ) as file:
            print(json.dumps(lista_hoteles), file=file)

        return reserva

    except json.JSONDecodeError as e:
        raise error(archivo_reservas) from e
```

In [24]:
```python
#Elemento Reservacciones -- acción cancelar reservación
#Definimos una función para cancelar una reserva ingresando 4 controles de busqueda

def borrar_reserva(indice_hotel, indice_huesped, fecha_entrada, fecha_salida):

    archivo_reservas = "reservas.json"

    try:
        datos_reserva = (
            f"{indice_hotel}_{indice_huesped}_"
            f"{fecha_entrada}_{fecha_salida}"
        )
        with open(archivo_reservas, 'r', encoding="utf-8") as file:
            lista = json.load(file)
            if len(lista) == 0:
                raise error_reserva(
                    datos_reserva
                )

            reservas = dict(
                (
                    (
                        f"{reser['ID_HOTEL']}_{reser['ID_HUESPED']}_"
                        f"{reser['FECHA_ENTRADA']}_{reser['FECHA_SALIDA']}"
                    ),
                    reser
                ) for reser in lista
            )
            if datos_reserva not in reservas:
                raise error_reserva(datos_reserva)

            lista.clear()
            del reservas[datos_reserva]
            lista.extend(list(reservas.values()))
            with open(archivo_reservas, "w", encoding="utf-8") as file:
                print(json.dumps(lista), file=file)

    except FileNotFoundError as exc:
        raise error_reserva(datos_reserva) from exc
    except json.JSONDecodeError as e:
        raise error(archivo_reservas) from e
```

In [25]:
```python
# Quinta Sección - Pruebas de operación
```

In [26]:

```
class Test_pruebas_general(unittest.TestCase):
    print("Iniciando pruebas")
    #Creamos una función para verificar directorios y conflictos de archivos
    def setUp(self):
        if os.path.exists("hoteles.json"):
            os.remove("hoteles.json")
        else:
            print("No se encontró el archivo")

        if os.path.exists("huespedes.json"):
            os.remove("huespedes.json")
        else:
            print("No se encontró el archivo")

        if os.path.exists("reservas.json"):
            os.remove("reservas.json")
        else:
            print("No se encontró el archivo")

    #Probamos la función "nuevo_hotel"
    def test_probar_nuevo_hotel(self):

        nombre = nuevo_hotel("FiestaInn")
        self.assertEqual(nombre['NOMBRE_HOTEL'], "FiestaInn",
                         'La función "nuevo_hotel" no funciona correctamente')

    #Verificamos la función de prueba
    def test_probar_nuevo_hotel2(self):

        nuevo_hotel("FiestaInn")
        nuevo_hotel("Riu")

        with open(
            "hoteles.json", 'r', encoding="utf-8"
        ) as file:
            lista_real = json.load(file)
            self.assertEqual(
                len(lista_real), 2, 'El test de "nuevo_hotel"no funciona correctamente'
            )

    #Probamos la función "borrar_hotel"
    def test_probar_borrar_hotel(self):

        correcto = True

        try:
            nuevo_hotel("One")
            borrar_hotel(1)
        except error_hotel:
            correcto = False

        self.assertEqual(correcto, True,
                         'La función "borrar_hotel" no funciona correctamente')

    #Verificamos la función de prueba
    def test_probar_borrar_hotel2(self):

        correcto = False
        try:
            nuevo_hotel("One")
            borrar_hotel(2)
        except error_hotel:
            correcto = True

        self.assertEqual(correcto, True,
                         'El test de "borrar_hotel" no funciona correctamente')
```

```
#Probamos la funcion "mostrar_hotel"
def test_probar_mostrar_hotel(self):

    nuevo_hotel("FiestaInn")
    prueba = mostrar_hotel(1)
    self.assertEqual(prueba['NOMBRE_HOTEL'], 'FiestaInn',
                     'La función "mostrar_hotel" no funciona correctamente')

#Verificamos la función de prueba
def test_probar_mostrar_hotel2(self):

    correcto = False
    try:
        nuevo_hotel("FiestaInn")
        mostrar_hotel(2)
    except error_hotel:
        correcto = True

    self.assertEqual(correcto, True,
                     'El test de "nuevo_hotel" no funciona correctamente')

#Probamos la funcion "modificar hotel"
def test_probar_modificar_hotel(self):

    nuevo_hotel("One")
    prueba = modificar_hotel(1, "Aranzasu")

    self.assertEqual(prueba['ID_HOTEL'], 1,
                     'La función "modificar hotel" tiene error en ID')
    self.assertEqual(prueba['NOMBRE_HOTEL'], "Aranzasu",
                     'La función "modificar hotel" tiene error en NOMBRE')

#Verificamos la función de prueba
def test_probar_modificar_hotel2(self):

    correcto = False
    try:
        nuevo_hotel("One")
        modificar_hotel(2, "Aranzasu")
    except error_hotel:
        correcto = True

    self.assertEqual(correcto, True,
                     'El test de "modificar_hotel" no funciona correctamente')

#Provamos la función "nuevo_huesped"
def test_probar_nuevo_huesped(self):

    prueba = nuevo_huesped("Carlos")
    self.assertEqual(prueba['NOMBRE_HUESPED'], 'Carlos',
                     'La función "nuevo_huesped" no funciona correctamente')

#Verificamos la función de prueba
def test_probar_nuevo_huesped2(self):

    nuevo_huesped("Carlos")
    nuevo_huesped("Ramon")

    with open("huespedes.json", 'r', encoding="utf-8") as file:
        lista = json.load(file)
        self.assertEqual(len(lista), 2, 'El test de "nuevo_huesped"no funciona correctamente')

#Probamos la función "borrar_huesped"
def test_probar_borrar_huesped(self):

    correcto = True
```

```
    try:
        nuevo_huesped("Carlos")
        borrar_huesped(1)
    except error_huesped:
        correcto = False

    self.assertEqual(correcto, True,
                     'La función "borrar_huesped" no funciona correctamente')

#Verificamos la función de prueba
def test_probar_borrar_huesped2(self):

    correcto = False
    try:
        nuevo_huesped("Carlos")
        borrar_huesped(2)
    except error_huesped:
        correcto = True

    self.assertEqual(correcto, True,
                     'El test de "borrar_huesped" no funciona correctamente')

#Verificamos la función "mostrar_huesped"
def test_probar_mostrar_huesped(self):
    nuevo_huesped("Carlos")
    prueba = mostrar_huesped(1)
    self.assertEqual(prueba['NOMBRE_HUESPED'], 'Carlos',
                     'la función "mostrar_huesped" no funciona correctamente')

#Verificamos la función de prueba
def test_probar_mostrar_huesped2(self):
    correcto = False
    try:
        nuevo_huesped("Carlos")
        mostrar_huesped(2)
    except error_huesped:
        correcto = True

    self.assertEqual(correcto, True,
                     'El test de "mostrar_huesped" no funciona correctamente')

#Verificamos la función "modificar_huesped"
def test_probar_modificar_huesped(self):

    nuevo_huesped("Carlos")
    prueba = modificar_huesped(1, "Eugenio")
    self.assertEqual(prueba['ID_HUESPED'], 1,
                     'La función "modificar_huesped" tiene error en ID')
    self.assertEqual(prueba['NOMBRE_HUESPED'], "Eugenio",
                     'La función "modificar_huesped" tiene error en NOMBRE')

#Verificamos la función de prueba
def test_probar_modificar_huesped2(self):

    correcto = False
    try:
        nuevo_huesped("Carlos")
        modificar_huesped(2, "Eugenio")
    except error_huesped:
        correcto = True

    self.assertEqual(correcto, True,
                     'El test "modificar_huesped" no funciona corresctamente')

#Verificamos la función "nueva_reserva"
def test_probar_nueva_reserva(self):
```

```
    hotel = nuevo_hotel("CancunInn")
    huesped = nuevo_huesped("Carlos")
    prueba = nueva_reserva(
        hotel['ID_HOTEL'], huesped['ID_HUESPED'],
        "2023-12-02", "2023-12-10"
    )
    self.assertEqual(prueba['FECHA_ENTRADA'], '2023-12-02',
                     'la funcion "nueva_reserva" no funciona correctamente')

#Verificamos la función de prueba
def test_probar_nueva_reserva2(self):

    hotel = nuevo_hotel("CancunInn")
    huesped = nuevo_huesped("Carlos")
    nueva_reserva(hotel['ID_HOTEL'], huesped['ID_HUESPED'],
        "2023-12-01", "2023-12-05"
    )
    nueva_reserva(
        hotel['ID_HOTEL'], huesped['ID_HUESPED'],
        "2023-12-05", "2023-12-10"
    )

    with open("reservas.json", 'r', encoding="utf-8") as file:
        lista = json.load(file)
        self.assertEqual(
            len(lista), 2, 'El test de "nueva_reserva" no funciona correctamente'
        )
#Probamos la función "borrar_reserva"
def test_probar_borrar_reserva(self):

    prueba = True
    try:
        hotel = nuevo_hotel("One")
        huesped = nuevo_huesped("Federico")
        elemento = nueva_reserva(
            hotel['ID_HOTEL'], huesped['ID_HUESPED'],
            "2023-12-01", "2023-12-10"
        )
        borrar_reserva(
            elemento['ID_HOTEL'], elemento['ID_HUESPED'],
            elemento['FECHA_ENTRADA'], elemento['FECHA_SALIDA']
        )
    except error_reserva:
        prueba = False

    self.assertEqual(prueba, True,
                     'La función "borrar_reserva" no funciona correctamente')

#Verificamos la función de prueba
def test_probar_borrar_reserva2(self):

    correcto = False
    try:
        hotel = nuevo_hotel("One")
        huesped = nuevo_huesped("Federico")
        elemento = nueva_reserva(
            hotel['ID_HOTEL'], huesped['ID_HUESPED'],
            "2023-12-01", "2023-12-10"
        )
        borrar_reserva(
            2, elemento['ID_HUESPED'],
            elemento['FECHA_ENTRADA'], elemento['FECHA_SALIDA']
        )
    except error_reserva:
        correcto = True

    self.assertEqual(correcto, True,
```

```
                        'El test de "borrar_reserva"no funciona correctamente')
    print("Terminando pruebas")
```

```
Iniciando pruebas
Terminando pruebas
```

In [27]:
```
if __name__ == '__main__':
    unittest.main()
```

```
E
======================================================================
ERROR: C:\Users\traba\AppData\Roaming\jupyter\runtime\kernel-03ed0810-2eed-489f-a92d-a04448a1dc31 (unittest.loader._FailedTest.C:\Users\traba\AppData\Roaming\jupyter\runtime\kernel-03ed0810-2eed-489f-a92d-a04448a1dc31)
----------------------------------------------------------------------
AttributeError: module '__main__' has no attribute 'C:\Users\traba\AppData\Roaming\jupyter\runtime\kernel-03ed0810-2eed-489f-a92d-a04448a1dc31'

----------------------------------------------------------------------
Ran 1 test in 0.000s

FAILED (errors=1)
An exception has occurred, use %tb to see the full traceback.

SystemExit: True
C:\ProgramData\anaconda3\Lib\site-packages\IPython\core\interactiveshell.py:3534: UserWarning: To exit: use 'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

In [ ]: