**1. Logging Guest Check-ins**

Adds a log entry whenever a guest checks in.

import datetime

def log\_checkin(func):

def wrapper(\*args, \*\*kwargs):

print(f"{datetime.datetime.now()} - Guest Checked In: {args[0]}")

return func(\*args, \*\*kwargs)

return wrapper

@log\_checkin

def check\_in(guest\_name):

print(f"Welcome, {guest\_name}! Your room is ready.")

check\_in("John Doe")

**2. Authentication for Hotel Manager**

Restricts access to manager-only functions.

def manager\_only(func):

def wrapper(\*args, \*\*kwargs):

user = kwargs.get("user", "guest")

if user != "manager":

print("Access Denied! Only managers can perform this action.")

return

return func(\*args, \*\*kwargs)

return wrapper

@manager\_only

def generate\_financial\_report(user="guest"):

print("Generating financial report...")

generate\_financial\_report(user="manager") # Allowed

generate\_financial\_report(user="guest") # Denied

**3. Applying Discounts for VIP Guests**

Automatically applies a discount if the guest is a VIP.

def vip\_discount(func):

def wrapper(guest, is\_vip=False):

price = func(guest, is\_vip)

if is\_vip:

price \*= 0.8 # 20% discount

print(f"VIP Discount Applied! New Price: ${price:.2f}")

return price

return wrapper

@vip\_discount

def room\_price(guest, is\_vip=False):

return 200 # Standard room price

room\_price("Alice", is\_vip=True)

room\_price("Bob", is\_vip=False)

**4. Automatic Tip Calculation for Room Service**

Adds a tip to the bill if not specified.

def add\_tip(func):

def wrapper(\*args, \*\*kwargs):

total = func(\*args, \*\*kwargs)

tip = total \* 0.1 # 10% tip

print(f"Adding Tip: ${tip:.2f}")

return total + tip

return wrapper

@add\_tip

def room\_service(order\_total):

return order\_total

print(f"Total Bill: ${room\_service(50):.2f}")

**5. Checking Room Availability**

Prevents booking if the room is unavailable.

available\_rooms = {"101": True, "102": False}

def check\_availability(func):

def wrapper(room\_number, \*args, \*\*kwargs):

if not available\_rooms.get(room\_number, False):

print(f"Room {room\_number} is not available.")

return

return func(room\_number, \*args, \*\*kwargs)

return wrapper

@check\_availability

def book\_room(room\_number, guest\_name):

print(f"Room {room\_number} booked for {guest\_name}")

book\_room("101", "Emma") # Success

book\_room("102", "Liam") # Denied

**6. Enforcing Dress Code in Hotel Restaurant**

Rejects guests not meeting the dress code.

def dress\_code\_required(func):

def wrapper(guest, attire):

if attire.lower() not in ["formal", "semi-formal"]:

print(f"Sorry {guest}, you must wear formal attire.")

return

return func(guest, attire)

return wrapper

@dress\_code\_required

def enter\_restaurant(guest, attire):

print(f"{guest} is allowed into the restaurant.")

enter\_restaurant("Charlie", "casual") # Denied

enter\_restaurant("Diana", "formal") # Allowed

**7. Automatic Check-out Reminder**

Sends a reminder to guests who have stayed beyond their checkout time.

import datetime

def checkout\_reminder(func):

def wrapper(guest, checkout\_time):

now = datetime.datetime.now().hour

if now > checkout\_time:

print(f"Reminder: {guest}, please check out!")

return func(guest, checkout\_time)

return wrapper

@checkout\_reminder

def stay\_duration(guest, checkout\_time):

print(f"{guest} has checked out at {checkout\_time}:00.")

stay\_duration("Oliver", 10) # Before time

stay\_duration("Sophia", 8) # Late checkout

**8. Sanitization of User Input for Hotel Booking**

Prevents SQL injection or unsafe characters.

import re

def sanitize\_input(func):

def wrapper(guest\_name):

safe\_name = re.sub(r'[^a-zA-Z\s]', '', guest\_name)

return func(safe\_name)

return wrapper

@sanitize\_input

def book\_hotel(guest\_name):

print(f"Hotel booked under the name: {guest\_name}")

book\_hotel("Robert'; DROP TABLE Guests;--") # SQL injection attempt

book\_hotel("Emily Rose") # Safe input

**9. Limiting Room Service Orders per Guest**

Restricts guests from ordering more than a set limit.

order\_count = {}

def limit\_orders(func):

def wrapper(guest):

if order\_count.get(guest, 0) >= 3:

print(f"Sorry {guest}, you have reached your order limit.")

return

order\_count[guest] = order\_count.get(guest, 0) + 1

return func(guest)

return wrapper

@limit\_orders

def order\_room\_service(guest):

print(f"Room service order placed for {guest}")

order\_room\_service("Lily")

order\_room\_service("Lily")

order\_room\_service("Lily")

order\_room\_service("Lily") # Denied

**10. Timing Execution for Hotel Services**

Measures the time taken to complete a task.

import time

def timing\_decorator(func):

def wrapper(\*args, \*\*kwargs):

start = time.time()

result = func(\*args, \*\*kwargs)

end = time.time()

print(f"Time taken: {end - start:.2f} seconds")

return result

return wrapper

@timing\_decorator

def housekeeping\_service():

time.sleep(2) # Simulating work

print("Housekeeping service completed.")

housekeeping\_service()