```
def is_valid_sudoku(board):
    def is_valid_group(group):
        nums = [num for num in group if num != "."]
        return len(nums) == len(set(nums))  # Check if all numbers are unique

    # Check rows and columns
    for i in range(9):
        if not is_valid_group(board[i]):  # Check row i
            print(f"Invalid row {i+1}")
            return False
        if not is_valid_group([board[j][i] for j in range(9)]):  # Check column i
            print(f"Invalid column {i+1}")
            return False

    # Check 3×3 subgrids
    for row in range(0, 9, 3):
        for col in range(0, 9, 3):
            subgrid = [board[r][c] for r in range(row, row+3) for c in range(col, col+3)]
            if not is_valid_group(subgrid):
                print(f"Invalid 3x3 grid at row {row+1}, column {col+1}")
                return False

    return True

# Example Sudoku Board (Valid)
sudoku_board = [
    ["5","3",".",".","7",".",".",".",".","."],
    ["6",".",".","1","9","5",".",".",".","."],
    [".","9","8",".",".",".",".",".","6","."],
    ["8",".",".",".","6",".",".",".",".","3"],
    ["4",".",".","8",".","3",".",".",".","1"],
    ["7",".",".",".",".","2",".",".",".",".","6"],
    [".","6",".",".",".",".",".",".","2","8","."],
    [".",".",".",".","4","1","9",".",".",".","5"],
    [".",".",".",".",".","8",".",".",".","7","9"]
]

# Example Invalid Sudoku Board (Duplicate in row)
invalid_sudoku_board = [
    ["5","3",".",".","7",".",".",".",".","."],
    ["6","5",".","1","9","5",".",".",".","."],  # Duplicate '5' in row 2
    [".","9","8",".",".",".",".",".","6","."],
    ["8",".",".",".","6",".",".",".",".","3"],
    ["4",".",".","8",".","3",".",".",".","1"],
    ["7",".",".",".",".","2",".",".",".",".","6"],
    [".","6",".",".",".",".",".",".","2","8","."],
    [".",".",".",".","4","1","9",".",".",".","5"],
    [".",".",".",".",".","8",".",".",".","7","9"]
]

print("Valid Board Check:", is_valid_sudoku(sudoku_board))  # Output: True
print("Invalid Board Check:", is_valid_sudoku(invalid_sudoku_board))  # Output: False wit
```

```
Valid Board Check: True
Invalid row 2
Invalid Board Check: False
```

```
#Task 40
from collections import Counter

def word_frequency(text):
    words = text.lower().split()  # Convert to lowercase and split into words
    word_count = Counter(words)   # Count occurrences using Counter
    return dict(word_count)       # Convert to dictionary and return

# Example Input
text = "Hello world! This is a test. Hello again, world!"

# Remove punctuation and count words
import re
cleaned_text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
result = word_frequency(cleaned_text)

# Output Result
print("Word Frequencies:", result)
```

```
Word Frequencies: {'hello': 2, 'world': 2, 'this': 1, 'is': 1, 'a': 1, 'test': 1, 'ag
```

```
#Task 41
def knapsack(weights, values, capacity):
    n = len(weights)
    dp = [[0] * (capacity + 1) for _ in range(n + 1)]  # DP table

    # Fill the DP table
    for i in range(1, n + 1):
        for w in range(capacity + 1):
            if weights[i - 1] <= w:
                # Maximize value: take or leave the item
                dp[i][w] = max(dp[i - 1][w], values[i - 1] + dp[i - 1][w - weights[i - 1]
            else:
                dp[i][w] = dp[i - 1][w]

    return dp[n][capacity]  # Maximum value that can be carried

# Example Input
weights = [2, 3, 4, 5]
values = [3, 4, 5, 6]
capacity = 5

# Output Result
print("Maximum Value:", knapsack(weights, values, capacity))
```

```
Maximum Value: 7
```

```
#Task 41
def merge_intervals(intervals):
    if not intervals:
        return []

    # Step 1: Sort intervals by start time
    intervals.sort(key=lambda x: x[0])

    merged = [intervals[0]]  # Initialize merged list with the first interval

    for start, end in intervals[1:]:
        last_end = merged[-1][1]  # Get the end of the last merged interval

        if start <= last_end:
            # Merge overlapping intervals by updating the end time
            merged[-1][1] = max(last_end, end)
        else:
            # Add non-overlapping interval
            merged.append([start, end])

    return merged

# Example Input
intervals = [[1, 3], [2, 6], [8, 10], [15, 18]]

# Output Result
print("Merged Intervals:", merge_intervals(intervals))
```

```
Merged Intervals: [[1, 6], [8, 10], [15, 18]]
```

```
#Task 43
def find_median_sorted_arrays(nums1, nums2):
    # Merge the two sorted arrays
    merged = sorted(nums1 + nums2)

    n = len(merged)

    # Find median
    if n % 2 == 1:
        return merged[n // 2]  # Odd length, return the middle element
    else:
        return (merged[n // 2 - 1] + merged[n // 2]) / 2  # Even length, return average o

# Example Input
nums1 = [1, 3]
nums2 = [2]

# Output Result
print("Median:", find_median_sorted_arrays(nums1, nums2))
```

```
Median: 2
```

```
#Task 44
def largest_rectangle_area(heights):
    stack = []
    max_area = 0
    heights.append(0)  # Sentinel to clear stack at end

    for i, h in enumerate(heights):
        while stack and h < heights[stack[-1]]:
            height = heights[stack.pop()]
            width = i if not stack else i - stack[-1] - 1
            max_area = max(max_area, height * width)
        stack.append(i)

    return max_area

def maximal_rectangle(matrix):
    if not matrix or not matrix[0]:
        return 0

    rows, cols = len(matrix), len(matrix[0])
    heights = [0] * cols
    max_area = 0

    for row in matrix:
        for j in range(cols):
            heights[j] = heights[j] + 1 if row[j] == '1' else 0  # Build histogram
        max_area = max(max_area, largest_rectangle_area(heights))

    return max_area

# Example Input
matrix = [
    ["1","0","1","0","0"],
    ["1","0","1","1","1"],
    ["1","1","1","1","1"],
    ["1","0","0","1","0"]
]

# Output Result
print("Maximal Rectangle Area:", maximal_rectangle(matrix))
```

```
Maximal Rectangle Area: 6
```

```
#Task 45
def max_subarray_sum(arr):
    max_sum = float('-inf')
    current_sum = 0

    for num in arr:
        current_sum += num
        max_sum = max(max_sum, current_sum)
        if current_sum < 0:
```

```
            current_sum = 0  # Reset if negative sum

    return max_sum

# Example Input
arr = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

# Output Result
print("Largest Sum Contiguous Subarray:", max_subarray_sum(arr))
```

```
Largest Sum Contiguous Subarray: 6
```

```
#Task 46
from collections import deque

def word_ladder(start_word, end_word, word_dict):
    word_dict = set(word_dict)  # Convert to set for O(1) lookups
    if end_word not in word_dict:
        return 0  # End word must be in dictionary

    queue = deque([(start_word, 1)])  # (word, transformation length)

    while queue:
        word, length = queue.popleft()

        if word == end_word:
            return length  # Found the shortest path

        # Try all possible one-letter changes
        for i in range(len(word)):
            for char in 'abcdefghijklmnopqrstuvwxyz':
                new_word = word[:i] + char + word[i+1:]

                if new_word in word_dict:
                    queue.append((new_word, length + 1))
                    word_dict.remove(new_word)  # Remove to prevent cycles

    return 0  # If no transformation is found

# Example Input
start = "hit"
end = "cog"
word_list = ["hot", "dot", "dog", "lot", "log", "cog"]

# Output Result
print("Shortest Transformation Sequence Length:", word_ladder(start, end, word_list))
```

```
Shortest Transformation Sequence Length: 5
```

```
#Task 6
import random
import json
```

```
import os

# Character Class
class Character:
    def __init__(self, name, health, attack, defense):
        self.name = name
        self.health = health
        self.attack = attack
        self.defense = defense

    def is_alive(self):
        return self.health > 0

    def take_damage(self, damage):
        self.health -= max(0, damage - self.defense)

# Player Class
class Player(Character):
    def __init__(self, name):
        super().__init__(name, health=100, attack=15, defense=5)
        self.inventory = []

    def pick_item(self, item):
        self.inventory.append(item)
        print(f"You picked up {item}!")

    def show_inventory(self):
        if self.inventory:
            print("Inventory:", ", ".join(self.inventory))
        else:
            print("Your inventory is empty.")

# Enemy Class
class Enemy(Character):
    def __init__(self, name, health, attack, defense):
        super().__init__(name, health, attack, defense)

# Game Functions
def combat(player, enemy):
    print(f"\n⚔️ A {enemy.name} appears!")

    while player.is_alive() and enemy.is_alive():
        action = input("\nChoose action: [A]ttack or [R]un: ").lower()
        if action == "a":
            damage = random.randint(5, player.attack)
            enemy.take_damage(damage)
            print(f"You hit the {enemy.name} for {damage} damage!")

            if enemy.is_alive():
                enemy_damage = random.randint(5, enemy.attack)
                player.take_damage(enemy_damage)
                print(f"The {enemy.name} hit you for {enemy_damage} damage!")

        elif action == "r":
            if random.random() > 0.5:
```

```
                print("You successfully ran away!")
                return
            else:
                print("You failed to escape!")

    if player.is_alive():
        print(f"\nYou defeated the {enemy.name}!")
    else:
        print("\nYou have been defeated... Game Over!")

def explore(player):
    print("\n🌍 Exploring the world...")
    event = random.choice(["enemy", "item", "nothing"])

    if event == "enemy":
        enemy = Enemy("Goblin", 50, 10, 2)
        combat(player, enemy)
    elif event == "item":
        item = random.choice(["Potion", "Sword", "Shield"])
        player.pick_item(item)
    else:
        print("Nothing happened.")

def save_game(player):
    data = {
        "name": player.name,
        "health": player.health,
        "attack": player.attack,
        "defense": player.defense,
        "inventory": player.inventory
    }
    with open("savegame.json", "w") as f:
        json.dump(data, f)
    print("Game saved!")

def load_game():
    if not os.path.exists("savegame.json"):
        return None
    with open("savegame.json", "r") as f:
        data = json.load(f)
    player = Player(data["name"])
    player.health = data["health"]
    player.attack = data["attack"]
    player.defense = data["defense"]
    player.inventory = data["inventory"]
    print("Game loaded!")
    return player

# Main Game Loop
def main():
    print("\n🎮 Welcome to the RPG Game!")

    if os.path.exists("savegame.json"):
        choice = input("Load saved game? (y/n): ").lower()
        if choice == "y":
```

```
            player = load_game()
        else:
            player_name = input("Enter your name: ")
            player = Player(player_name)
    else:
        player_name = input("Enter your name: ")
        player = Player(player_name)

    while player.is_alive():
        print(f"\n🛡️ {player.name} - Health: {player.health}")
        action = input("\nChoose action: [E]xplore, [I]nventory, [S]ave, [Q]uit: ").lower

        if action == "e":
            explore(player)
        elif action == "i":
            player.show_inventory()
        elif action == "s":
            save_game(player)
        elif action == "q":
            print("Thanks for playing!")
            break
        else:
            print("Invalid choice!")

if __name__ == "__main__":
    main()
```

```
🎮 Welcome to the RPG Game!
Enter your name: Harshi

🛡️ Harshi - Health: 100

Choose action: [E]xplore, [I]nventory, [S]ave, [Q]uit: E

🌍 Exploring the world...

⚔️ A Goblin appears!

Choose action: [A]ttack or [R]un: A
You hit the Goblin for 7 damage!
The Goblin hit you for 8 damage!

Choose action: [A]ttack or [R]un: A
You hit the Goblin for 14 damage!
The Goblin hit you for 9 damage!

Choose action: [A]ttack or [R]un: A
You hit the Goblin for 8 damage!
The Goblin hit you for 7 damage!

Choose action: [A]ttack or [R]un: R
You successfully ran away!

🛡️ Harshi - Health: 91
```

```
Choose action: [E]xplore, [I]nventory, [S]ave, [Q]uit: Q
Thanks for playing!
```

Start coding or generate with AI.