The Problem

Context: You have a box containing 10 six-sided dice.

- **Defective Die (1):** One die is defective with the following probabilities for its faces:
    - Side 1: 10%
    - Side 2: 10%
    - Side 3: 10%
    - Side 4: 20%
    - Side 5: 20%
    - Side 6: 30%
- **Normal Dice (9):** The other nine dice are fair, with the probability of rolling any side being 1/6.

Task: You randomly select **two** dice from the box and roll them. Calculate the **expected sum** of the values you will roll. Round your answer to the nearest thousandth.

The Solution

This problem hinges on a core concept in probability: **Expected Value** and the **Linearity of Expectation**.

Step 1: Calculate the expected value of the defective die (E_defective).
The expected value is the sum of each outcome multiplied by its probability.
E_defective = (1 * 0.10) + (2 * 0.10) + (3 * 0.10) + (4 * 0.20) + (5 * 0.20) + (6 * 0.30)
E_defective = 0.1 + 0.2 + 0.3 + 0.8 + 1.0 + 1.8
E_defective = 4.2

Step 2: Calculate the expected value of a normal (fair) die (E_normal).
E_normal = (1 + 2 + 3 + 4 + 5 + 6) * (1/6)
E_normal = 21 / 6
E_normal = 3.5

Step 3: Calculate the expected value of a single die drawn randomly from the box (E_single_draw).

When you draw one die, you have a 1/10 chance of picking the defective one and a 9/10 chance of picking a normal one. We can use the law of total expectation:
E_single_draw = P(Defective) * E_defective + P(Normal) * E_normal
E_single_draw = (1/10) * 4.2 + (9/10) * 3.5
E_single_draw = 0.42 + 3.15
E_single_draw = 3.57

Step 4: Calculate the expected sum of TWO dice drawn from the box.
This is where the **Linearity of Expectation** is crucial. It states that E[X + Y] = E[X] + E[Y], regardless of whether X and Y are independent.

- Let X1 be the result of the first die you roll.
- Let X2 be the result of the second die you roll.

The expected value of the first die, E[X1], is the value we just calculated: **3.57**.
Since the process is identical for the second die (it's also drawn from the same box of 10), its expected value, E[X2], is also **3.57**.

Therefore, the expected sum is:
Expected Sum = E[X1] + E[X2]
Expected Sum = 3.57 + 3.57
Expected Sum = 7.14

Rounding to the nearest thousandth, the final answer is **7.140**.

Knowledge Skeleton: Expected Value

Here is a more structured breakdown of the underlying topic for interview preparation.

Part 1: The Core Concept (Theoretical Foundations)

- **What is it?** The Expected Value (or expectation, E[X]) of a random variable X is the long-run average value of the variable over many repeated experiments. It's a weighted average of all possible outcomes, where the weights are the probabilities of those outcomes. For a discrete variable, it's calculated as `E[X] = Σ [x * P(X=x)]`.

- **Why does it matter?** Expected value is a fundamental concept in data science and machine learning:
    - **Loss Functions:** The goal of training most models is to minimize the *expected* loss over the entire data distribution.
    - **Business Metrics:** Calculating Customer Lifetime Value (CLV) or expected revenue from a marketing campaign relies on expected value.
    - **Algorithms:** It's the core of Reinforcement Learning (maximizing expected future rewards) and foundational to understanding concepts like bias (the difference between the expected value of an estimator and the true value).
- **Key Mathematical Principle: Linearity of Expectation.** For any two random variables X and Y, **E[X + Y] = E[X] + E[Y].** This property is extremely powerful because it holds true **even if the variables are not independent**. This is what allowed us to solve the problem above so elegantly without considering the three complex scenarios (Normal+Normal, Normal+Defective, Defective+Normal).

Part 2: The Interview Gauntlet (Theoretical Questions)

- **Conceptual Understanding:**
    i. "What is the definition of expected value? How is it different from the mean of a dataset?"
    ii. "Can the expected value of a variable be a value that the variable itself can never take? Give an example." (Answer: Yes, the expected value of a single fair die roll is 3.5).
    iii. "What is the 'Law of the Unconscious Statistician' (LOTUS)?" (Answer: It's a theorem for calculating E[g(X)] without finding the distribution of g(X) first).
- **Intuition & Trade-offs:**
    i. "Do two random variables need to be independent for the expectation of their sum to be the sum of their expectations? Why is this property so useful in practice?"
    ii. "Describe a scenario where relying solely on the expected value to make a decision could be misleading." (Answer: When the distribution is highly skewed, like lottery winnings or investment returns. The median might be a better measure of central tendency).

iii. "You are A/B testing a new 'buy' button. How would you frame the decision of which button is better using the concept of expected value?" (Answer: The 'better' button is the one that maximizes the expected revenue per user, calculated as `(probability of click) * (average purchase value given a click)`).

- **Troubleshooting & Edge Cases:**
    i. "How would you estimate the expected value of a variable if you only have a sample of data and not the true probabilities?" (Answer: You calculate the sample mean. The sample mean is an unbiased estimator of the population expectation).
    ii. "What happens to the expected value calculation for a continuous variable?" (Answer: The summation is replaced by an integral: `E[X] = ∫ x * f(x) dx`, where f(x) is the probability density function).

Part 3: The Practical Application (Code & Implementation)

In Python, you rarely calculate theoretical expected value from a formula unless you're in a specific simulation or probability problem. More commonly, you **estimate** the expected value from a sample of data. The `numpy.mean()` or `pandas.Series.mean()` functions are the tools for this.

The sample mean is a powerful estimator for the true expected value because of the **Law of Large Numbers**, which states that as your sample size grows, the sample mean will converge to the true expected value.

```python
import numpy as np

# A sample of 10,000 rolls from our defective die
# Note: p must sum to 1
outcomes = [1, 2, 3, 4, 5, 6]
probabilities = [0.1, 0.1, 0.1, 0.2, 0.2, 0.3]

# Simulate 10,000 rolls
rolls = np.random.choice(outcomes, size=10000, p=probabilities)

# Estimate the expected value by calculating the sample mean
estimated_expectation = np.mean(rolls)
theoretical_expectation = 4.2

print(f"Theoretical Expected Value: {theoretical_expectation}")
print(f"Estimated Expected Value from 10,000 samples: {estimated_expectation:.4f}")
# Output will be very close to 4.2
```

Part 4: The Code Challenge (Practical Questions)

Challenge: Write a Python function that *simulates* the original problem. The function should take the number of trials `N` as an input. In each trial, it should:

1. Define the population of 10 dice (1 defective, 9 normal).
2. Randomly draw two dice *without replacement*.
3. Roll each of the two chosen dice to get their values.
4. Sum the values.
   The function should run `N` trials and return the average sum. Verify that for a large `N` (e.g., 100,000), the result approaches the theoretical answer of **7.140**.

Answer:

```python
import numpy as np

def roll_die(die_type):
    """Rolls a single die of a given type ('normal' or 'defective')."""
    if die_type == 'defective':
        return np.random.choice([1, 2, 3, 4, 5, 6], p=[0.1, 0.1, 0.1, 0.2, 0.2, 0.3])
    else: # 'normal'
        return np.random.choice([1, 2, 3, 4, 5, 6])

def simulate_dice_sum(num_trials=100000):
    """
    Simulates drawing and rolling two dice from the box N times
    and returns the average sum.
    """
    # Define the population of dice in the box
    dice_box = ['defective'] + ['normal'] * 9

    total_sum = 0
    for _ in range(num_trials):
        # 1. Draw two dice without replacement
        chosen_dice_indices = np.random.choice(10, size=2, replace=False)
        die1_type = dice_box[chosen_dice_indices[0]]
        die2_type = dice_box[chosen_dice_indices[1]]

        # 2. Roll each die and sum the results
        roll1 = roll_die(die1_type)
        roll2 = roll_die(die2_type)
        trial_sum = roll1 + roll2

        # 3. Add to the total
        total_sum += trial_sum

    # 4. Calculate the average sum over all trials
    average_sum = total_sum / num_trials
    return average_sum

# Run the simulation
simulation_result = simulate_dice_sum(num_trials=100000)
```

```python
print(f"Theoretical Expected Sum: 7.140")
print(f"Simulated Average Sum over 100,000 trials: {simulation_result:.3f}")

# Expected output:
# Theoretical Expected Sum: 7.140
# Simulated Average Sum over 100,000 trials: 7.141 (or a value very close to 7.140)
```