📚 Module 10: Ratios - Complete Notes

🎯 What You'll Learn

In this module, you'll master **ratios** — expressing relationships between quantities. This powers proportional allocation, scaling, and comparative analysis.

📖 Concept Explained (Like a YouTube Video)

The Basics

A **ratio** shows how two quantities relate. It says "for every X of this, there are Y of that."

```
Ratio 3:2 means:
"For every 3 units of A, there are 2 units of B"

Visual:
A: 🟦🟦🟦
B: 🟨🟨

If we double it (scale by 2):
A: 🟦🟦🟦🟦🟦🟦
B: 🟨🟨🟨🟨

Still 3:2! Same ratio, different amounts.
```

Key Concepts

```
Part-to-Part:  3:2  (A compared to B)
Part-to-Whole: 3:5  (A compared to total A+B)

Simplify like fractions:
6:4 → 3:2 (divide both by GCD = 2)
15:10 → 3:2 (divide both by GCD = 5)
```

💻 Programming Connection

Code Examples

```python
# Example 1: Simplify Ratio

import math

def simplify_ratio(a, b):
    """Simplify ratio to lowest terms"""
    gcd = math.gcd(a, b)
    return (a // gcd, b // gcd)

print(simplify_ratio(6, 4))      # (3, 2)
```

```python
print(simplify_ratio(100, 25))   # (4, 1)
print(simplify_ratio(15, 9))     # (5, 3)
```

```python
# Example 2: Distribute by Ratio

def distribute_by_ratio(total, *parts):
    """Distribute total according to ratio"""
    total_parts = sum(parts)
    return [total * p // total_parts for p in parts]

# Split $100 in ratio 3:2
print(distribute_by_ratio(100, 3, 2))   # [60, 40]

# Split 120 items in ratio 2:3:5
print(distribute_by_ratio(120, 2, 3, 5))  # [24, 36, 60]
```

```python
# Example 3: Check Equivalent Ratios

def are_equivalent(ratio1, ratio2):
    """Check if two ratios are equivalent"""
    # Cross multiply: a:b = c:d if a*d = b*c
    a, b = ratio1
    c, d = ratio2
    return a * d == b * c

print(are_equivalent((3, 2), (6, 4)))   # True
print(are_equivalent((3, 2), (4, 3)))   # False
```

```python
# Example 4: Scale Ratio to Target Total

def scale_ratio(ratio, target_total):
    """Scale ratio parts to sum to target"""
    current_total = sum(ratio)
    scale = target_total / current_total
    return [int(r * scale) for r in ratio]

# Make 3:2 add up to 100
print(scale_ratio([3, 2], 100))   # [60, 40]
```

🧪 SDET/Testing Application

```python
# SDET Scenario: Calculate Pass/Fail Ratio

def calculate_ratio(passed, failed):
    """Calculate simplified pass:fail ratio"""
    import math
```

```python
    if failed == 0:
        return f"{passed}:0 (all passed)"

    gcd = math.gcd(passed, failed)
    return f"{passed // gcd}:{failed // gcd}"

print(calculate_ratio(80, 20))   # "4:1"
print(calculate_ratio(75, 25))   # "3:1"
print(calculate_ratio(90, 10))   # "9:1"
```

```python
# SDET Scenario: Distribute Test Data

def allocate_test_data(total, train_ratio, test_ratio):
    """Allocate data based on ratio"""
    total_parts = train_ratio + test_ratio
    train = total * train_ratio // total_parts
    test = total - train  # Remaining to avoid rounding issues

    return {"train": train, "test": test}

# 80:20 split of 1000 items
result = allocate_test_data(1000, 80, 20)
print(result)  # {'train': 800, 'test': 200}
```

🎓 Practice Problems

Problem 1: Easy 🟢

Challenge: Simplify the ratio 45:30.

Problem 2: Medium 🟡

Challenge: Split $240 among three people in the ratio 2:3:5.

Problem 3: Application 🔴

Scenario: Your test pass:fail ratio is 4:1. Total tests = 100.
Challenge: How many passed? How many failed?

🔑 Key Takeaways

✅ **Ratio = Relative comparison** — Not absolute values
✅ **Simplify using GCD** — Same as fractions
✅ **Part-to-Whole** — For calculating portions
✅ **Distribute** — multiply by (part/sum_of_parts)

💾 *Save as:* `Module_10_Ratios.md`