📚 Module 13: Time-Distance-Speed - Complete Notes

🎯 What You'll Learn

In this module, you'll master **rate relationships** — the formula triangle that applies to throughput, bandwidth, and any rate-based calculation.

📖 Concept Explained (Like a YouTube Video)

The Core Formula

```
        Distance
           △
          / \
         /   \
        /     \
    Speed × Time

Cover what you want:
- Distance = Speed × Time
- Speed = Distance / Time
- Time = Distance / Speed
```

This Pattern Is EVERYWHERE

```
Testing:    Tests = Tests/Minute × Minutes
Bandwidth:  Data = Rate × Time
Throughput: Requests = RPS × Seconds
API Calls:  Cost = Price/Token × Tokens
```

💻 Programming Connection

Code Examples

```python
# Example 1: Basic Calculations

def calculate_speed(distance, time):
    """Speed = Distance / Time"""
    if time == 0:
        return float('inf')
    return distance / time

def calculate_distance(speed, time):
    """Distance = Speed × Time"""
    return speed * time

def calculate_time(distance, speed):
    """Time = Distance / Speed"""
```

```
    if speed == 0:
        return float('inf')
    return distance / speed

# Examples
print(calculate_speed(100, 2))     # 50 km/h
print(calculate_distance(60, 3))   # 180 km
print(calculate_time(150, 50))     # 3 hours
```

```
# Example 2: Throughput Calculation

def calculate_throughput(requests, time_seconds):
    """Calculate requests per second"""
    return requests / time_seconds

def estimate_completion(total_requests, rps):
    """Estimate time to complete all requests"""
    seconds = total_requests / rps

    if seconds < 60:
        return f"{seconds:.1f} seconds"
    elif seconds < 3600:
        return f"{seconds/60:.1f} minutes"
    else:
        return f"{seconds/3600:.1f} hours"

# 1000 requests in 10 seconds
rps = calculate_throughput(1000, 10)
print(f"Throughput: {rps} req/s")  # 100.0 req/s

# Time for 1 million requests at 1000 rps
print(estimate_completion(1_000_000, 1000))  # 16.7 minutes
```

```
# Example 3: Average Speed Trap!

def average_speed(distances, times):
    """Average speed = TOTAL distance / TOTAL time"""
    total_distance = sum(distances)
    total_time = sum(times)
    return total_distance / total_time

# Trip 1: 100km in 2h (50 km/h)
# Trip 2: 100km in 4h (25 km/h)
# Average is NOT (50 + 25) / 2 = 37.5!

avg = average_speed([100, 100], [2, 4])
print(f"Average speed: {avg:.1f} km/h")  # 33.3 km/h (not 37.5!)
```

```python
# Example 4: Unit Conversion

def format_duration(seconds):
    """Convert seconds to human-readable format"""
    if seconds < 60:
        return f"{seconds:.0f}s"
    elif seconds < 3600:
        mins = seconds // 60
        secs = seconds % 60
        return f"{mins:.0f}m {secs:.0f}s"
    else:
        hours = seconds // 3600
        mins = (seconds % 3600) // 60
        return f"{hours:.0f}h {mins:.0f}m"

print(format_duration(45))     # 45s
print(format_duration(125))    # 2m 5s
print(format_duration(3725))   # 1h 2m
```

🧪 SDET/Testing Application

```python
# SDET Scenario: API Load Test Planning

def plan_load_test(target_requests, max_rps, safety_buffer=1.2):
    """Plan load test duration"""
    raw_seconds = target_requests / max_rps
    buffered_seconds = raw_seconds * safety_buffer

    return {
        "target_requests": target_requests,
        "max_rps": max_rps,
        "minimum_duration": format_duration(raw_seconds),
        "recommended_duration": format_duration(buffered_seconds),
        "buffer": f"{(safety_buffer-1)*100:.0f}% safety margin"
    }

result = plan_load_test(100_000, 500)
print(f"Recommended: {result['recommended_duration']}")
```

```python
# SDET Scenario: Test Execution Estimate

def estimate_test_run(total_tests, avg_test_duration_sec):
    """Estimate total test run time"""
    total_seconds = total_tests * avg_test_duration_sec

    return {
        "total_tests": total_tests,
        "avg_per_test": f"{avg_test_duration_sec}s",
```

```
        "total_duration": format_duration(total_seconds),
        "tests_per_minute": 60 / avg_test_duration_sec
    }

result = estimate_test_run(500, 3)
print(f"Total time: {result['total_duration']}")  # 25m 0s
print(f"Rate: {result['tests_per_minute']:.0f} tests/min")
```

🎓 Practice Problems

Problem 1: Easy 🟢

Challenge: 100 tests in 5 minutes. What's the tests/second rate?

Problem 2: Medium 🟡

Challenge: API handles 500 RPS. Time for 1 million requests?

Problem 3: Application 🔴

Scenario: Current test suite: 1000 tests, 2 sec each, runs sequentially.
Challenge: How long does it take? If you parallelize across 4 runners?

⚠️ Common Mistakes

❌ **Mistake 1**: Averaging speeds directly
✅ **Fix**: Total distance / Total time

❌ **Mistake 2**: Forgetting unit consistency
✅ **Fix**: Convert everything to same units first

🔑 Key Takeaways

✅ **Speed = Distance / Time**
✅ **Pattern applies everywhere** — Throughput, bandwidth, cost
✅ **Average speed** — Total distance / Total time (not average of speeds)
✅ **Inverse relationship** — Faster speed means less time

💾 *Save as:* `Module_13_Time_Distance_Speed.md`

Phase 2 Complete! 🚀