Notebook 2.4 - Integration Challenge

Management Science - Preparing for CEO at Bean Counter!

Introduction

You've climbed the ranks:

- Assistant Manager → Mastered functions for standardized operations
- Regional Manager → Conquered dictionaries for data management
- Operations Director → Optimized with sorting and selection

Now the CEO wants to see if you can integrate all these skills. This is your chance to make an impression before potentially taking over as CEO of Bean Counter!

Today's Challenge: Build an integrated system that uses functions, dictionaries, and sorting together to solve real business problems.

> **i** How to Use This Tutorial
>
> This is a shorter recap session. Each exercise integrates multiple concepts you've learned.

Section 1 - Warm-Up: Functions Meet Dictionaries

Let's start by combining functions with dictionaries, the foundation of business logic and data management.

```
# Combining what you've learned
def calculate_store_score(store):
    """Calculate a performance score for a store"""
    # Weight different metrics
    sales_score = store["daily_sales"] / 100  # Scale to reasonable number
    rating_score = store["rating"] * 20  # Rating is 1-5, scale to 20-100
    efficiency_score = store["customers_per_staff"] * 2

    total_score = sales_score + rating_score + efficiency_score
    return round(total_score, 1)

# Test with a store
test_store = {
    "name": "Downtown",
    "daily_sales": 4500,
    "rating": 4.7,
    "customers_per_staff": 65
}
```

```
score = calculate_store_score(test_store)
print(f"{test_store['name']} score: {score}")
```

```
Downtown score: 269.0
```

Exercise 1.1 - Performance Calculator

Create a function `evaluate_store` that:

1. Takes a store dictionary as input (see above)
2. Calculates if the store is meeting targets
3. Returns a tuple: `(is_successful, performance_message)`

Success criteria:

- Daily sales > 4000 AND
- Customer rating >= 4.5 AND
- Staff efficiency > 50 customers/staff

> **Tip**
>
> The returned `is_successful` is just a boolean (True or False), while the performance message should be a string with feedback for the store.

```
# YOUR CODE BELOW
def evaluate_store(store):
    """Evaluate if a store meets Bean Counter standards"""
    # Check all three criteria

    # Return (success_boolean, message)
```

```
# Test your evaluation function
store1 = {"name": "Plaza", "daily_sales": 5000, "rating": 4.8,
"customers_per_staff": 75}
success1, msg1 = evaluate_store(store1)
assert success1 == True, "Plaza should be successful"

store2 = {"name": "Beach", "daily_sales": 3500, "rating": 4.6,
"customers_per_staff": 60}
success2, msg2 = evaluate_store(store2)
assert success2 == False, "Beach should not be successful (low sales)"

print("Great! Your evaluation function works perfectly!")
```

Section 2 - The Power of Integration

Now let's combine all three concepts: functions process the data, dictionaries store it, and sorting helps us make decisions.

```
# Integration example: Finding top performers
def calculate_efficiency(store):
    """Calculate operational efficiency"""
    return store["revenue"] / store["costs"]

# Sample stores
stores = [
    {"name": "Airport", "revenue": 50000, "costs": 35000},
    {"name": "Downtown", "revenue": 45000, "costs": 28000},
    {"name": "Beach", "revenue": 30000, "costs": 22000}
]

# Add efficiency to each store
for store in stores:
    store["efficiency"] = calculate_efficiency(store)

# Sort by efficiency
ranked = sorted(stores, key=lambda x: x["efficiency"], reverse=True)

print("Efficiency Rankings:")
for i, store in enumerate(ranked, 1):
    print(f"{i}. {store['name']}: {store['efficiency']:.2f}")
```

```
Efficiency Rankings:
1. Downtown: 1.61
2. Airport: 1.43
3. Beach: 1.36
```

Exercise 2.1 - Complete Store Ranking System

Build a system that:

1. Calculates a composite score for each store using a function
2. Adds this score to each store dictionary
3. Ranks stores from best to worst
4. Returns the name of the best store

Composite score = (revenue/1000) + (rating * 10) - (complaints * 2)

```
def calculate_composite_score(store):
    """Calculate composite score for ranking"""
    # YOUR CODE BELOW

    return score

def find_best_store(stores_list):
    """Find the best store based on composite score"""
```

```
    # YOUR CODE BELOW
    # 1. Add composite score to each store

    # 2. Sort by composite score (highest first)

    # 3. Return the name of the best store

    return best_store_name

# Test data
stores = [
    {"name": "Plaza", "revenue": 45000, "rating": 4.7, "complaints": 5},
    {"name": "Station", "revenue": 38000, "rating": 4.9, "complaints": 2},
    {"name": "Airport", "revenue": 52000, "rating": 4.4, "complaints": 12}
]

best = find_best_store(stores)
print(f"The best performing store is: {best}")
```

```
# Test your ranking system
assert best == "Station", f"Best store should be Station, got {best}"

# Verify scores were calculated correctly
for store in stores:
    if store["name"] == "Plaza":
        assert store["composite_score"] == 82.0, "Plaza score incorrect"
    elif store["name"] == "Station":
        assert store["composite_score"] == 83.0, "Station score incorrect
(should be highest)"
    elif store["name"] == "Airport":
        assert store["composite_score"] == 72.0, "Airport score incorrect"

print("Excellent! Your integrated ranking system works perfectly!")
print("Station wins with the best composite score!")
print("You're ready to become the CEO of Bean Counter!")
```

Conclusion

Congratulations! You're ready to become CEO of Bean Counter!

You've successfully integrated:

- Functions - For calculations and business logic
- Dictionaries - For structured data management
- Sorting - For optimization and decision-making

Your complete toolkit demonstrates:

- Evaluating store performance with custom metrics
- Ranking and comparing complex business entities
- Optimizing operational decisions with data

Remember:

- Functions make your code reusable and maintainable
- Dictionaries organize complex business data effectively
- Sorting and selection help you make optimal choices
- Combining these tools creates powerful business solutions

What's Next: You've completed the Python foundations! In Lecture 3, you'll explore advanced data science tools with NumPy and Pandas. These powerful libraries will help you analyze larger datasets and tackle even more complex management science challenges.

Solutions

You will likely find solutions to most exercises online. However, I strongly encourage you to work on these exercises independently without searching explicitly for the exact answers to the exercises. Understanding someone else's solution is very different from developing your own. Use the lecture notes and try to solve the exercises on your own. This approach will significantly enhance your learning and problem-solving skills.

Remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities. If you encounter difficulties, review the lecture materials, experiment with different approaches, and don't hesitate to ask for clarification during class discussions.

Bibliography