Notebook 2.2 - Dictionaries at Bean Counter

Management Science - Regional Manager of Multiple Locations!

Introduction

Welcome to your new role as Regional Manager at Bean Counter!

Congratulations on another promotion!

Your standardized functions from the Assistant Manager role were so successful that the CEO has promoted you to Regional Manager! You're now responsible for overseeing 10 Bean Counter locations across the city.

The Challenge: Managing 10 locations means tracking large amounts of data:

- Each location has different inventory levels
- Sales vary by neighborhood
- Barista teams have different sizes
- Equipment needs vary by store size

Using simple variables or lists isn't enough anymore. You need a better way to organize all this information!

Your Task: Understand dictionaries - Python's way of storing structured data with labels. Think of them as digital filing cabinets where everything has a clear label and is easy to find.

In this tutorial, we'll learn how dictionaries help you manage complex business data across all Bean Counter locations.

> **i** How to Use This Tutorial
>
> Cells marked with "YOUR CODE BELOW" expect you to write your own code. Test blocks will verify your solutions.

Section 1 - Dictionary Basics for Location Management

As Regional Manager, you need to track detailed information about each Bean Counter location. Dictionaries are perfect for this - they store data with descriptive labels (keys) instead of position numbers.

```
# A dictionary for one Bean Counter location
downtown_store = {
```

```
    "name": "Bean Counter Downtown",
    "manager": "Sarah Chen",
    "daily_sales": 4500.00,
    "staff_count": 8,
    "coffee_beans_kg": 45.5
}

# Accessing data is intuitive with keys
print(f"Location: {downtown_store['name']}")
print(f"Manager: {downtown_store['manager']}")
print(f"Today's sales: ${downtown_store['daily_sales']}")
```

```
Location: Bean Counter Downtown
Manager: Sarah Chen
Today's sales: $4500.0
```

Dictionary anatomy:

- Curly braces `{}` create a dictionary
- Keys (like "name", "manager") are labels for your data
- Values are the actual data (like "Sarah Chen", 4500.00)
- Access values using square brackets with the key: `dict['key']`

> Lists vs Dictionaries
>
> - Lists are like numbered storage boxes - you access items by position (0, 1, 2...)
> - Dictionaries are like labeled filing folders - you access items by name ("sales", "inventory"...)
> - Use dictionaries when you want to describe something with multiple attributes!

Exercise 1.1 - Create a Location Profile

Create a dictionary called `airport_store` for the Bean Counter airport location with:

- name: "Bean Counter Airport"
- manager: "James Wilson"
- daily_sales: 6200.00
- staff_count: 12
- coffee_beans_kg: 62.0

```
# YOUR CODE BELOW
airport_store = {
    # Add the key-value pairs here
}
```

```
# Test your location profile
assert airport_store["name"] == "Bean Counter Airport", "Store name should
be 'Bean Counter Airport'"
```

```
assert airport_store["manager"] == "James Wilson", "Manager should be
'James Wilson'"
assert airport_store["daily_sales"] == 6200.00, "Daily sales should be
6200.00"
assert airport_store["staff_count"] == 12, "Staff count should be 12"
assert airport_store["coffee_beans_kg"] == 62.0, "Coffee beans should be
62.0 kg"
print("Perfect! You've created your first location profile!")
```

Exercise 1.2 - Update Inventory

The airport store just received a delivery! Update the dictionary:

1. Add 25 kg to the current coffee_beans_kg
2. Update the manager to “Maria Garcia” (James got promoted!)
3. Add a new key “last_inspection” with value “2024-01-15”

> **Updating Dictionaries**
>
> - Update existing values: `dict['key'] = new_value`
> - Add new keys: `dict['new_key'] = value`
> - Modify values using math: `dict['number'] = dict['number'] + 10`

```
# Assuming airport_store exists from previous exercise
airport_store = {
    "name": "Bean Counter Airport",
    "manager": "James Wilson",
    "daily_sales": 6200.00,
    "staff_count": 12,
    "coffee_beans_kg": 62.0
}

# YOUR CODE BELOW
# Update coffee beans (add 25 kg)

# Update manager

# Add last_inspection
```

```
# Test your updates
assert airport_store["coffee_beans_kg"] == 87.0, "Coffee beans should now
be 87.0 kg"
assert airport_store["manager"] == "Maria Garcia", "Manager should be Maria
Garcia"
assert airport_store["last_inspection"] == "2024-01-15", "Should have
last_inspection date"
print("Excellent inventory management! The airport store is fully
updated!")
```

Section 2 - Lists of Dictionaries for Multiple Locations

Managing 10 Bean Counter locations means working with multiple dictionaries. We can organize them in a list!

```
# List of dictionaries - each dictionary is one store
bean_counter_locations = [
    {
        "id": 1,
        "neighborhood": "Downtown",
        "daily_sales": 4500,
        "needs_restock": False
    },
    {
        "id": 2,
        "neighborhood": "Airport",
        "daily_sales": 6200,
        "needs_restock": True
    },
    {
        "id": 3,
        "neighborhood": "University",
        "daily_sales": 3800,
        "needs_restock": False
    }
]

# Access specific stores and their data
print(f"First location: {bean_counter_locations[0]['neighborhood']}")
print(f"Second location sales: ${bean_counter_locations[1]
['daily_sales']}")
```

```
First location: Downtown
Second location sales: $6200
```

⚠ Common Mistake

When accessing data from a list of dictionaries, remember the two-step process:

1. First, get the dictionary from the list: `list[index]`
2. Then, get the value from the dictionary: `list[index]['key']`

Exercise 2.1 - Build Your Regional Network

Create a list called `regional_stores` containing 2 Bean Counter locations. Each store dictionary should have:

- location_name (string)
- monthly_revenue (number)
- manager_name (string)
- flagship (boolean - True for one store, False for others)

```
# YOUR CODE BELOW
regional_stores = [
    # Add 2 store dictionaries here
]
```

```
# Test your regional network
assert len(regional_stores) == 2, "Should have exactly 2 stores"
assert all("location_name" in store for store in regional_stores), "Each
store needs location_name"
assert all("monthly_revenue" in store for store in regional_stores), "Each
store needs monthly_revenue"
assert all("manager_name" in store for store in regional_stores), "Each
store needs manager_name"
assert all("flagship" in store for store in regional_stores), "Each store
needs flagship status"
assert sum(store["flagship"] for store in regional_stores) >= 1, "At least
one store should be flagship"
print("Great work! Your regional network is established!")
```

Section 3 - Looping Through Store Data

As Regional Manager, you need to analyze data across all locations. Let's loop through dictionaries to find insights!

```
# Analyzing multiple stores
stores = [
    {
        "name": "Downtown",
        "sales": 4500,
        "rating": 4.8
    },
    {
        "name": "Airport",
        "sales": 6200,
        "rating": 4.5
    },
    {
        "name": "University",
        "sales": 3800,
        "rating": 4.9
    }
]

# Calculate total sales across all stores
total_sales = 0
for store in stores:
    total_sales = total_sales + store["sales"]
    print(f"{store['name']}: ${store['sales']} (Rating: {store['rating']}
★)")

print(f"\nTotal regional sales: ${total_sales}")
```

```
Downtown: $4500 (Rating: 4.8★)
Airport: $6200 (Rating: 4.5★)
University: $3800 (Rating: 4.9★)

Total regional sales: $14500
```

> **Accessing Dictionary Items**
>
> When looping through a list of dictionaries:
>
> - The loop variable (like `store`) represents one complete dictionary
> - Access values using keys: `store['key_name']`
> - You can use the values in calculations, conditions, or function calls

Exercise 3.1 - Regional Performance Analysis

Write code to analyze the `performance_data` list:

1. Calculate the total customers served across all stores
2. Find the average customer satisfaction score
3. Count how many stores served over 500 customers

```
performance_data = [
    {"store": "Downtown", "customers": 450, "satisfaction": 4.7},
    {"store": "Airport", "customers": 680, "satisfaction": 4.4},
    {"store": "University", "customers": 520, "satisfaction": 4.8},
    {"store": "Mall", "customers": 380, "satisfaction": 4.6},
    {"store": "Station", "customers": 590, "satisfaction": 4.5}
]

total_customers = 0
total_satisfaction = 0
busy_stores = 0  # Stores with >500 customers
# YOUR CODE BELOW
for store in performance_data:
    # Add to total customers

    # Add to satisfaction sum

    # Check if this is a busy store


# Print your results
#
```

```
# Test your analysis
assert total_customers == 2620, f"Total customers should be 2620, got
{total_customers}"
assert round(average_satisfaction, 2) == 4.60, f"Average satisfaction
should be 4.60, got {round(average_satisfaction, 2)}"
```

```
assert busy_stores == 3, f"Should have 3 busy stores, got {busy_stores}"
print(f"Total customers: {total_customers}")
print(f"Average satisfaction: {average_satisfaction}")
print(f"Stores serving >500 customers: {busy_stores}")
print("Excellent analysis! You have a clear picture of regional
performance!")
```

Section 4 - Filtering Store Data

Regional Managers often need to find stores that meet specific criteria - which ones need supplies? Which are underperforming?

```
# Finding stores that need attention
all_stores = [
    {"name": "Downtown", "inventory_kg": 15, "daily_sales": 4500},
    {"name": "Airport", "inventory_kg": 8, "daily_sales": 6200},
    {"name": "University", "inventory_kg": 22, "daily_sales": 3800},
    {"name": "Mall", "inventory_kg": 5, "daily_sales": 4100}
]

# Find stores with low inventory (less than 10 kg)
low_inventory_stores = []
for store in all_stores:
    if store["inventory_kg"] < 10:
        low_inventory_stores.append(store["name"])

print("Stores needing coffee beans:", low_inventory_stores)
```

```
Stores needing coffee beans: ['Airport', 'Mall']
```

Exercise 4.1 - Identify Priority Stores

Find stores that need immediate attention. Create two lists:

1. `restock_needed`: Store names with inventory below 20 kg
2. `low_performers`: Store names with daily revenue below 1000

```
inventory_data = [
    {"name": "Plaza", "coffee_kg": 45, "milk_liters": 30, "daily_revenue":
1850},
    {"name": "Station", "coffee_kg": 12, "milk_liters": 8, "daily_revenue":
920},
    {"name": "Park", "coffee_kg": 18, "milk_liters": 15, "daily_revenue":
1100},
    {"name": "Beach", "coffee_kg": 7, "milk_liters": 22, "daily_revenue":
780},
    {"name": "Downtown", "coffee_kg": 35, "milk_liters": 18,
"daily_revenue": 2200}
]
```

```
# YOUR CODE BELOW
restock_needed = []
low_performers = []

for store in inventory_data:
    # Check if coffee inventory is low (below 20 kg)

    # Check if daily revenue is low (below 1000)
```

```
# Test your filtering
assert set(restock_needed) == {"Station", "Park", "Beach"}, f"Restock list
should be Station, Park, Beach"
assert set(low_performers) == {"Station", "Beach"}, f"Low performers should
be Station and Beach"
print(f"Stores needing restock: {restock_needed}")
print(f"Low performing stores: {low_performers}")
print("Perfect! You've identified the stores that need immediate
attention!")
```

Section 5 - Dictionaries with Functions

Let's combine your function skills from the Assistant Manager role with dictionaries to create powerful analysis tools!

```
def calculate_store_bonus(store_data):
    """Calculate performance bonus based on store metrics"""
    base_bonus = 1000

    # Add bonus for high sales
    if store_data["monthly_sales"] > 100000:
        base_bonus = base_bonus + 500

    # Add bonus for good ratings
    if store_data["customer_rating"] >= 4.5:
        base_bonus = base_bonus + 300

    return base_bonus

# Test the function
store = {
    "name": "Downtown",
    "monthly_sales": 120000,
    "customer_rating": 4.7
}
bonus = calculate_store_bonus(store)
print(f"{store['name']} earned a ${bonus} bonus!")
```

```
Downtown earned a $1800 bonus!
```

Exercise 5.1 - Store Performance Report Generator

Create a function `generate_store_report` that takes a store dictionary and returns a performance summary string.

The function should:

1. Calculate profit: revenue - costs
2. Calculate profit margin: (profit / revenue) * 100
3. Determine status: "Excellent" if margin > 30%, "Good" if > 20%, else "Needs Improvement"
4. Return a formatted string with the store name, margin, and status

```
# YOUR CODE BELOW
def generate_store_report(store):
    # Calculate profit margin

    # Determine performance status

    # Return formatted report string
```

```
# Test your report generator
test_store1 = {"name": "Plaza", "revenue": 50000, "costs": 30000}
report1 = generate_store_report(test_store1)
assert "40.0%" in report1, "Plaza should have 40.0% margin"
assert "Excellent" in report1, "Plaza should be Excellent"

test_store2 = {"name": "Station", "revenue": 40000, "costs": 34000}
report2 = generate_store_report(test_store2)
assert "15.0%" in report2, "Station should have 15.0% margin"
assert "Needs Improvement" in report2, "Station needs improvement"

print("Fantastic! Your reporting system gives clear insights into store
performance!")
```

Conclusion

Congratulations! You've successfully built a regional data management system for Bean Counter!

You've learned:

- Dictionaries - Storing structured data with descriptive labels (keys and values)
- Accessing & Updating - Getting and modifying dictionary values using keys
- Lists of Dictionaries - Managing multiple entities with consistent structure
- Looping Through Dictionaries - Analyzing data across all stores
- Filtering Dictionary Data - Finding stores that meet specific criteria
- Functions with Dictionaries - Creating powerful analysis and reporting tools

Your Bean Counter regional management system can now:

- Track detailed information for each location
- Analyze performance across all stores
- Identify locations needing supplies or support
- Generate automated performance reports

Remember:

- Use curly braces `{}` to create dictionaries
- Access values with square brackets and keys: `dict['key']`
- Dictionaries are perfect for describing things with multiple attributes
- Lists of dictionaries let you manage many similar entities
- Loop through dictionaries to aggregate data or find patterns
- Combine dictionaries with functions for sophisticated business logic

What's Next: Your data management skills have caught the CEO's attention! In the next tutorial, you'll be promoted to Operations Director, where you'll learn about sorting and optimization - finding the best options among many choices. You'll optimize scheduling, resource allocation, and discover how to always pick the best option for Bean Counter!

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