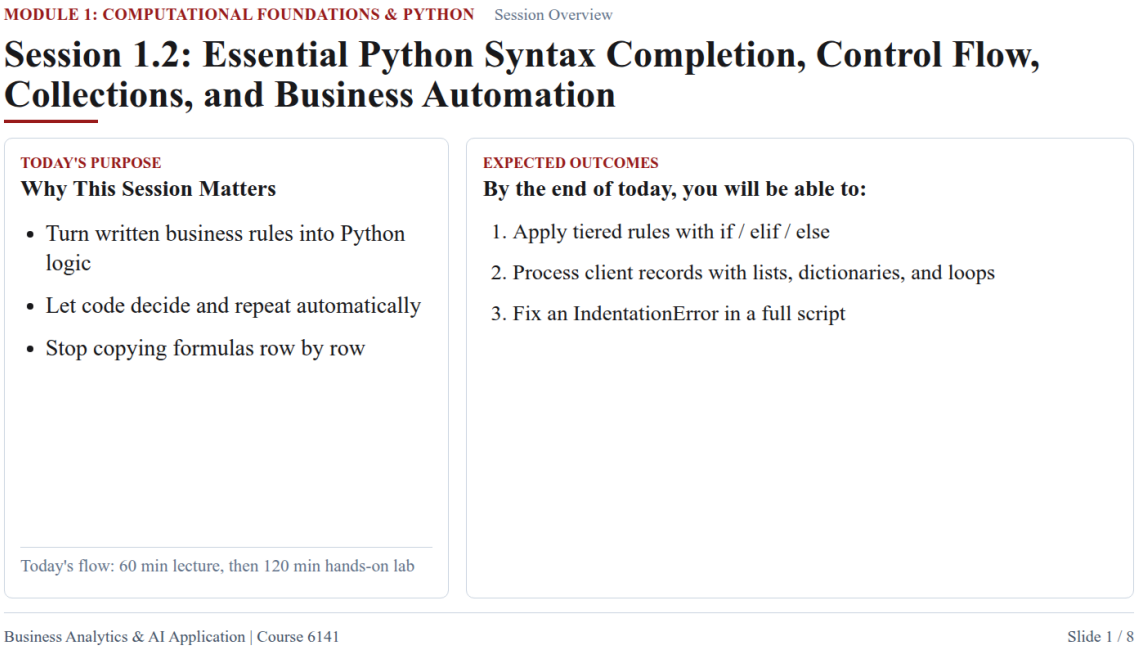
Session 1.2: Essential Python Syntax Completion: Control Flow, Collections, and Business Automation

Course 6141: Business Analytics & AI Application | Reading for Week 2



Schedule: Week 2 | In-person classes (1:2 Delivery Ratio: 60 Min Lecture + 120 Min Lab)

Session Purpose

- Turn written business rules into Python logic
- Let code decide and repeat automatically
- Stop copying formulas row by row

Expected Learning Outcomes (By the end of this session, you will be able to)

1. Apply tiered rules with if / elif / else
2. Process client records with lists, dictionaries, and loops
3. Fix an IndentationError in a full script

Keywords (Core Programming Concepts)

- `Boolean Expression`: A logical comparison that evaluates strictly to `True` or `False`, used by computers to test commercial conditions (such as `annual_spend >= 50000.0`).
- `Conditional Branching (else)`: Directing code execution down distinct logical pathways depending on operational thresholds, eliminating complex spreadsheet nested logic.
- `List Collection`: An ordered, mutable sequence of business records enclosed in square brackets (e.g., `daily_orders = [120, 85, 94, 142]`), indexed sequentially starting from zero.
- `Dictionary Collection`: A key-value associative data structure enclosed in curly braces (e.g., `{"client_id": "CUST-101", "annual_spend_USD": 145000.0}`), enabling instant data retrieval by descriptive business keys rather than numeric indices.
- `Definite Iteration (for loop)`: A programmatic control structure that automatically iterates over every record in a collection, executing identical business rules across thousands of entries in milliseconds.

Core Programming & Computational Foundations

- Eliminating Fragile Nested Spreadsheets: In Excel, multi-tier decision policies require nested formulas such as `=IF(C2>100000, "Platinum", IF(C2>50000, "Gold", IF(C2>20000, "Silver", "Standard")))`. These formulas quickly become unreadable, impossible to audit, and vulnerable to formula drift. In Python, conditional branching (`if-elif-else`) structures business rules into clean vertical indentation blocks that mirror managerial decision trees.
- Deterministic Repetition over Human Fatigue: When processing transactional ledgers with thousands of rows, human analysts experience fatigue, eye strain, and copy-paste errors. A Python `for` loop executes repetitive calculations across 10 records or 1,000,000 records with identical mathematical precision and zero cognitive degradation.
- Real-World Entity Modeling with Dictionaries: Modeling complex commercial entities as structured computational objects. A single customer account is captured as a dictionary of labeled attributes (name, industry, spend, risk score), while an enterprise client portfolio is structured as an iterable list of dictionaries.
- Indentation as Syntax Structure (The Whitespace Discipline): Unlike older programming languages that use curly brackets `{}` or `BEGIN/END` tokens, Python enforces clean code formatting through whitespace indentation (4 spaces). An `IndentationError` occurs when code blocks are misaligned, serving as an automated guardrail for code readability.

Hands-On Coding Micro-Examples

- *Micro-Example 1 (Conditional Customer Tiering):* Evaluating client annual spend to determine customer loyalty tiers and discount rates:

```
annual_spend = 82000.0

if annual_spend >= 100000.0:
    tier = "Platinum"
    discount_rate = 0.15
elif annual_spend >= 50000.0:
    tier = "Gold"
```

```
    discount_rate = 0.10
elif annual_spend >= 20000.0:
    tier = "Silver"
    discount_rate = 0.05
else:
    tier = "Standard"
    discount_rate = 0.0

print(f"Customer Tier: {tier} | Approved Discount: {discount_rate * 100:.0f}%")
```

- *Micro-Example 2 (Iterative Revenue Accumulation):* Looping through a list of daily store sales to calculate cumulative revenue and average daily performance:

```
daily_sales = [4200.0, 5100.0, 3900.0, 6200.0, 7800.0, 8400.0, 4900.0]
total_revenue = 0.0

for sale in daily_sales:
    total_revenue += sale

avg_daily_sales = total_revenue / len(daily_sales)
print("Total Weekly Revenue (USD):", round(total_revenue, 2))
print("Average Daily Sales (USD):", round(avg_daily_sales, 2))
```

- *Micro-Example 3 (Dictionary Client Profile & Safe Retrieval):* Storing commercial account attributes in a dictionary and computing available credit limits:

```
client = {
    "client_id": "CUST-104",
    "company_name": "Apex Logistics",
    "annual_spend_USD": 215000.0,
    "credit_limit_USD": 75000.0,
    "outstanding_balance_USD": 23400.0
}

available_credit = client["credit_limit_USD"] - client["outstanding_balance_USD"]
print(f"{client['company_name']} Available Credit: ${available_credit:,.2f}")
```

Lab Practice Dataset

- File Link: `data/session_01_02_dataset.csv`
- Standard 10-row commercial client dataset used for batch automation and loop exercises:
  - `Customer_ID`: Unique client identification code (e.g., `CUST-101`).
  - `Industry_Segment`: Commercial vertical (Retail, Technology, Healthcare, Finance, Manufacturing).
  - `Annual_Spend_USD`: Total historical purchase volume in U.S. Dollars.
  - `Order_Count`: Total orders completed in the fiscal year.
  - `Avg_Order_Value_USD`: Average transaction size in U.S. Dollars.

- `Payment_Terms_Days_Count`: Invoiced credit settlement period (30, 60, 90 days).
- `Risk_Score_Count`: Internal risk assessment index (1 to 100).

Official Python Documentation & References

- Python 3.12 Official Tutorial: More Control Flow Tools (if Statements and for Loops).
- Python 3.12 Official Tutorial: Data Structures (Lists, Dictionaries, and Tuples).
- PEP 8: Style Guide for Python Code (Indentation and Whitespace Standards).

Step-by-Step Lab Protocol (Exercise 1: Guided Practice - Client Portfolio Automation)

- *Step 1: Create Practice File:* Launch Google Antigravity IDE and create a new script named `session_1_2_automation.py` in your working directory.
- *Step 2: Build a Client Portfolio List:* Define an iterable list of customer dictionaries containing audited client records from `data/session_01_02_dataset.csv`:

```
clients = [
    {"id": "CUST-101", "segment": "Retail", "spend": 145000.0, "risk": 15},
    {"id": "CUST-102", "segment": "Technology", "spend": 82000.0, "risk": 22},
    {"id": "CUST-103", "segment": "Healthcare", "spend": 31000.0, "risk": 38},
    {"id": "CUST-104", "segment": "Finance", "spend": 215000.0, "risk": 12},
    {"id": "CUST-105", "segment": "Manufacturing", "spend": 64000.0, "risk": 45}
]
```

- *Step 3: Automate Tier Assignment and Credit Multiplier via for Loop:* Write a `for` loop that iterates over each client, applies conditional branching based on `spend`, and computes an approved credit limit:

```
for c in clients:
    # Tiering logic based on annual purchase volume
    if c["spend"] >= 100000.0 and c["risk"] < 30:
        c["tier"] = "Platinum"
        c["credit_limit"] = c["spend"] * 0.40
    elif c["spend"] >= 50000.0 and c["risk"] < 50:
        c["tier"] = "Gold"
        c["credit_limit"] = c["spend"] * 0.25
    else:
        c["tier"] = "Standard"
        c["credit_limit"] = c["spend"] * 0.10

    print(f"Client {c['id']} ({c['segment']}): Tier={c['tier']} | Credit
    ↪  Limit=${c['credit_limit']:,.2f}")
```

- *Step 4: Execute and Verify Output in Terminal:* Run the script in Panel `[4]` (Terminal) using `$ python session_1_2_automation.py`. Confirm that all 5 client accounts are evaluated and printed with proper credit limits.

Step-by-Step Applied Coding Practice (Exercise 2+: Hands-On Programming & AI Debugging)

- *Coding Challenge:* In Google Antigravity IDE, create a new script named `session_1_2_practice.py` to automate a multi-tier commercial commission payout pipeline.
- *Task 1 (Tiered Commission Functionality):* A sales team records quarterly sales volumes: `sales_reps = [{"name": "Alice", "revenue": 142000.0}, {"name": "Bob", "revenue": 68000.0}, {"name": "Charlie", "revenue": 35000.0}]`. Write a loop calculating commission payouts: `12%` for revenue over `$100,000`, `8%` for revenue between `$50,000` and `$100,000`, and `5%` for revenue below `$50,000`.
- *Task 2 (Deliberate IndentationError & AI Diagnosis):* Intentionally misalign one line inside the `if` block (e.g., adding 2 extra spaces or removing indentation). Run the script in Panel `[4]` (Terminal), observe the red `IndentationError: unexpected indent`, capture the terminal traceback with `Windows Key + Shift + S`, paste the image into Panel `[3]` (AI Agent Chat) with `Ctrl + V`, and review how the AI explains whitespace alignment.
- *Task 3 (Dictionary Summary Output):* Correct the indentation, append the calculated `commission_payout` back into each rep's dictionary, and print a formatted compensation summary with zero syntax errors. Verify that the script terminates with return code 0.