Below is the reformatted content for "Day 2: Python Control Flow – Decisions & Loops" based on your requirements. Code snippets are enclosed in code boxes, tables are structured with proper rows and columns, and deep explanations, prompts, and other narrative content are formatted differently for clarity. The date and time have been updated to 07:16 PM IST, Monday, June 23, 2025, as per the latest system update.

👩‍🏫 Day 2: Python Control Flow – Decisions & Loops (Trainer Script for Tech & Data Analysts)

🕒 Duration: 60–75 minutes

🎯 Ideal for: Data analysts, QA testers, junior developers, beginner automation engineers

🗓️ Date and Time: June 23, 2025, 07:16 PM IST

🎯 Session Objective

👩‍🏫 "Welcome back to Day 2 of our Python journey! Yesterday, we saw how to collect, store, and display data. But real-world applications need more than just storage—they need to respond to data. Today we'll dive into something extremely useful for analysts and developers alike—control flow. This is how we make Python make decisions and perform actions repeatedly. By the end of this session, you'll confidently:

- Implement conditional logic using if, elif, and else
- Apply comparison and logical operators to business rules
- Automate repetitive checks and processing with loops
- Control loops with break and continue
- Build logic-driven automation using a real-world support ticket example"

❓ **Engagement Question:** Think of one repetitive task or decision-making step you often do in Excel or SQL. What if Python could do it for you?

🧠 Part 1: Conditional Statements (if, elif, else)

Deep Explanation:

Conditional statements let your code choose what to do depending on the data. Think of a conditional like a decision gate. If the condition is true, the gate opens and the code runs. If not, Python looks for the next condition.

Code Snippet:

```text
sales = 98000
if sales >= 100000:
    print("Excellent performance")
elif sales >= 75000:
    print("Good job")
elif sales >= 50000:
    print("Needs improvement")
else:
    print("Poor performance")
```

Insight: This is just like writing a formula in Excel to classify customers by revenue.

Real-World Tech Use Cases:

- Dashboard Classification – Flagging KPIs like bounce rate or churn risk
- Email Campaign Filters – Targeting high-spending customers
- Quality Checks – Detecting faulty data or inconsistent reports

Trainer Tip: Ask learners to think: “How would you write logic to classify users based on number of app logins in the last 30 days?”

🔍 Part 2: Comparison and Logical Operators

Deep Explanation:

These operators are the language of decision-making. They let you compare values and combine conditions.

Table: Overview

Operator	Meaning	Example
==	Equal to	status == "Approved"
!=	Not equal to	region != "West"
>	Greater than	price > 500
<	Less than	attempts < 3
>=	Greater or equal	score >= 90
<=	Less or equal	quantity <= 100

Logical Operators:

- and: Both must be true
- or: At least one must be true
- not: Flips True to False and vice versa

Code Snippet:

```text
age = 28
location = "NY"
if age >= 25 and location == "NY":
    print("Send local NY event invite")
```

Analyst Application:

- Cross-checking user activity + subscription type
- Validating record completeness + flag status

Prompt for Learners: Write a condition to check if a transaction is over ₹10,000 and made via credit card.

🔁 Part 3: for Loops – Repeating Actions Over Data

Deep Explanation:

Loops allow you to take one action and repeat it across a list of items. In tech terms: “Apply this logic to every row, every record, or every user.”

Code Snippet:

```text
departments = ["Finance", "HR", "IT"]
for dept in departments:
    print("Processing:", dept)
```

Code Snippet:

```text
for i in range(1, 4):
    print("Attempt number:", i)
```

Tech Use Cases:

- Generating performance reports per department
- Sending emails to every customer in a filtered list
- Looping through test cases or log entries

Ask Learners: How could a for loop help you automate a daily report or check?

⏳ Part 4: while Loops – Repeat Until a Condition is Met

Deep Explanation:

Unlike for, a while loop doesn't know how many times it'll repeat. It keeps running until the condition becomes false. Think of it like checking a status every minute until a job is complete.

Code Snippet:

```text
confirmation = ""
while confirmation != "yes":
    confirmation = input("Do you approve this change? (yes/no): ")
```

Avoiding Infinite Loops:

Always make sure the loop eventually stops.

Code Snippet:

```text
while True:
    print("Stuck forever!")
```

Avoid unless you use break or a timeout.

Use Case:

- Waiting for valid input
- Polling a server for job completion
- Auto-retrying failed database syncs

🧯 Part 5: Loop Control – break and continue

Deep Explanation:

Sometimes, while looping, we want to skip something (continue) or stop altogether (break). Think of break like a fire alarm—it stops everything.

Code Snippet: Break Example

```text
records = ["valid", "valid", "error"]
for record in records:
    if record == "error":
        print("Stop: Faulty record")
        break
    print("Processing:", record)
```

Code Snippet: Continue Example

```text
emails = ["abc@example.com", "", "def@example.com"]
for email in emails:
    if email == "":
        continue
    print("Email sent to:", email)
```

Real Case:

- Stop processing a report if a critical metric is missing
- Skip empty values while looping over a CSV file

Prompt: Think of a task where you would skip or halt when hitting a specific condition.

🎮 Part 6: Real-World Mini Project – Flagging Open High-Priority Tickets

Scenario:

You are reviewing a list of support tickets. If the ticket is open and high priority, it must be escalated.

Code Snippet:

```text
tickets = [
    {"id": 101, "priority": "high", "status": "open"},
    {"id": 102, "priority": "low", "status": "closed"},
    {"id": 103, "priority": "high", "status": "closed"},
    {"id": 104, "priority": "high", "status": "open"}
]

for ticket in tickets:
    if ticket["priority"] == "high" and ticket["status"] == "open":
        print(f"🚨 Ticket {ticket['id']} needs immediate attention!")
```

Key Learnings:

- Looping through data
- Combining and logic
- Using string formatting for output

Practice Prompt: Can you modify this to count how many tickets were escalated?

✅ Summary & Best Practices for Analysts

Today's Learnings:

- Decision logic using if, elif, else
- Comparing values and combining rules with and, or, not
- Automating with for and while loops
- Controlling behavior with break, continue
- Building scripts that mirror real job tasks like ticket filtering

Best Practices:

- Keep loops and conditions clean and readable
- Use clear variable names (status, priority, threshold)
- Always test loop logic with small datasets first

Challenge Before Day 3: Try creating a script that flags any transaction over ₹10,000 made by a user from Delhi. Bonus: Count how many such users exist.

Coming Up Tomorrow: Functions in Python – how to organize and reuse your logic like a pro!

This format separates code into distinct boxes, structures tables with proper alignment, and uses a different style for explanations and prompts, making it easy to read and distinguish between content types. Let me know if you need further refinements!