Session 3 Overview: Full Operations and Growth

Session 3 involves automating the processing of entire business days, handling dozens of customers sequentially, tracking performance metrics, and implementing upgrade systems that drive long-term progression.

Daily Operations

Process complete business days automatically, looping through all customers, executing transactions, and summarizing results. Test purchase logic becomes production code handling dozens of transactions per day.

Performance Tracking

Count successful sales versus total customers, track daily revenue, identify best-selling items. These metrics inform strategic decisions.

Upgrade Systems

Implement three distinct upgrade paths—coffee machines boost customer count, manager training improves prices, decorations attract more visitors. Each upgrade costs money but provides permanent benefits.

This session introduces systems thinking—understanding how multiple mechanics interact to create complexity. Upgrades cost money, which comes from sales, which require inventory, which costs money to purchase. Balancing spending on upgrades versus inventory is key.

For Extended learners, Session 3 highlights object-oriented design. An Upgrade base class can define common behavior (cost calculation, level tracking) while subclasses implement specific effects. This polymorphism facilitates adding new upgrade types.

Tasks 1-2: Processing Complete Business Days

Task 1: Daily Customer Generation

Combine customer count and customer generation functions into a complete daily customer generation system. This function determines the number of customers, creates that number of customer objects, and returns them as a list for processing.

Normal: Function takes day number, returns list of customer dicts/tuples

Extended: Shop method returns list of Customer objects

Task 2: Process All Customers

Implement the automation to process the shop's operations. Loop through every customer in the day's list, process their purchase attempt, and accumulate statistics. Track three key metrics: total customers, successful sales, and total revenue earned.

Generate

Create full customer list

Iterate

Process each customer

Accumulate

Track sales and revenue

Summarize

Return daily results

Daily Revenue Mathematics

total_revenue = Σ (selling_price for each successful sale)

sales_count = Σ (1 for each successful purchase)

conversion_rate = sales_count / total_customers

Example: If 10 customers visit, 7 buy items averaging $3.50 each, then revenue = $24.50 and conversion = 70%

The process_customers function involves batch processing. This requires robust error handling, clear state management, and reliable tracking. If one customer's purchase fails, the loop must continue processing remaining customers without corruption.

Tasks 3-4: Daily Summary and Integration

Task 3 creates a reporting system for simulation results. After processing a full day of customers, display a clear summary of daily activity. The summary provides feedback on performance, identifying issues such as stock depletion or pricing.

Customer Traffic

Total visitors and successful purchases

Financial Results

Total revenue earned, average sale price

Inventory Impact

Items sold, items that ran out of stock

Performance Indicators

Conversion rate, comparison to previous day (optional)

Example Daily Summary

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
   DAY 5 RESULTS
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

Customers Today: 12
Successful Sales: 9
Revenue: $27.30
Conversion Rate: 75%

Items Sold:
  Coffee: 5
  Tea: 2
  Pastry: 2

Missed Sales: 3
  (Out of stock: 2)
  (Price too high: 1)
```

Task 4: Full Day Integration

Add a "Run business day" option to your main menu that orchestrates the complete daily cycle:

1. Open the store
2. Generate customers
3. Process all customers
4. Display daily summary
5. Close the store
6. Increment day counter

This single menu choice executes the entire business cycle automatically.

Task 4's integration combines existing functions into a cohesive daily cycle. When "Run business day" is selected, the system generates customers, processes transactions, manages inventory, and displays a summary. This automation streamlines multi-day simulation execution.

For Extended learners, implement this as a **run_business_day()** method that encapsulates the entire sequence. The method should be atomic—if something goes wrong midway, the state shouldn't be corrupted. Consider whether the method should return the daily summary object for external logging or analysis, or whether summaries should be stored internally in a daily_stats list.

Tasks 5-6: The Upgrade System

Upgrades introduce strategic decision-making by allowing investment in permanent improvements. Players must decide between spending money on inventory or on upgrades.

Manager Level

Improves selling prices by 10% per level

Coffee Machine

Increases daily customer count

Decoration Level

Increases shop appeal, attracting more visitors

Future Upgrades

Framework supports adding new upgrade types

Task 5: Upgrade Variables

Create variables for each upgrade level, all starting at 0:

- coffee_machine_level
- manager_level (already exists from Session 1)
- decoration_level

Extended: Store as Shop attributes or create Upgrade objects

Task 6: Coffee Machine Upgrade

Implement the first upgrade function:

- Calculate upgrade cost
- Check if shop has enough money
- Increase coffee_machine_level
- Deduct cost from money
- Display confirmation

Example cost formula: **cost = 10 × 1.5level**

Exponential Cost Formula

cost = base_cost × growth_factorcurrent_level

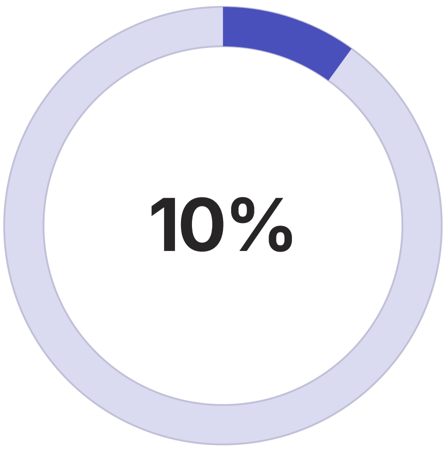
Example with base_cost=10, growth_factor=1.5:

Level 0→1: $10.00, Level 1→2: $15.00

The coffee machine's effect on customer count can be calculated using a formula such as: **base_customers + (coffee_machine_level × 2)**. This formula adds 2 additional customers per day per upgrade level.

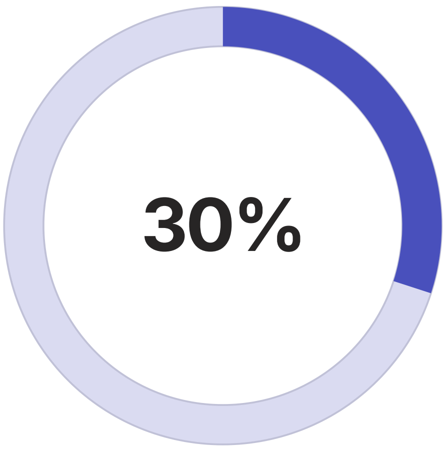
Tasks 7-8: Manager and Decoration Upgrades

Task 7 integrates manager level into the upgrade system. The manager level variable and its use in price calculations exist from Session 1. This task adds the ability to upgrade it by spending money. Each manager level increases selling prices by 10%, which increases profit margin on sales.



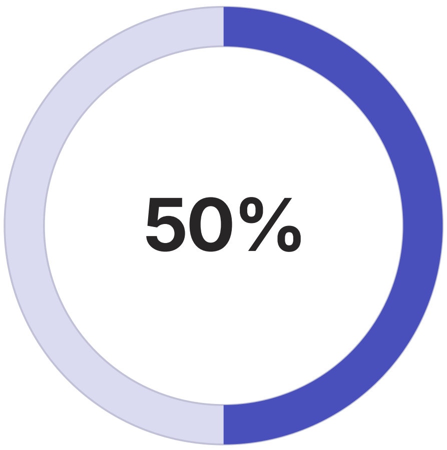
Per-Level Price Increase

Manager level increases selling price



Level 3 Example

Items sell for 30% more than base price



Level 5: Doubles Profit Margin

Level 5 provides a 50% price increase, doubling profit margin.

Manager Upgrade Impact

Manager level affects all transactions. For 100 items sold per week at $3 base price:

- **Level 0:** $3.00 each = $300 revenue
- **Level 2:** $3.60 each = $360 revenue (+$60)
- **Level 5:** $4.50 each = $450 revenue (+$150)

Percentage increase applies universally.

Decoration Upgrade Effect

Task 8 adds `decoration_level` to influence customer count. Customer count is determined by:

base = 5 + day

coffee_bonus = coffee_machine_level × 2

decoration_bonus = decoration_level × 1

total = base + coffee_bonus + decoration_bonus

Example Day 10, both at level 3: 15 + 6 + 3 = 24 customers

Create upgrade functions for both manager and decoration. Follow the coffee machine upgrade pattern: check cost, verify money, increase level, deduct payment, and confirm. Use the exponential cost formula with different base costs; for example, manager upgrades could use a $15 base, and decoration an $8 base.

Extended: Upgrade Class Hierarchy

Create an **Upgrade** base class with attributes (name, level, base_cost, growth_factor) and methods (calculate_cost, apply_upgrade). Implement the following subclasses:

- **CoffeeMachineUpgrade** - modifies customer count calculation
- **ManagerUpgrade** - modifies price calculation
- **DecorationUpgrade** - modifies customer count calculation

Each subclass implements an **apply_effect()** method to modify the appropriate shop calculation. This design supports adding new upgrades.

Tasks 9-10: Upgrade Integration

Task 9 makes upgrades accessible by adding menu options for each upgrade type. Your main simulation loop should offer choices such as "Upgrade coffee machine ($22.50)", "Upgrade manager ($33.75)", and "Upgrade decoration ($15.00)". Displaying the next upgrade cost in the menu provides necessary information for upgrade decisions.

1. View Status
2. Buy Items
3. Run Business Day
4. Upgrade Coffee Machine
5. Upgrade Manager
6. Upgrade Decoration
7. Quit

Task 10: Enhanced Status Display

Update your status screen to show all upgrade levels. Consider adding explanatory text about what each upgrade does.

Example Enhanced Status

```
═══════════════════════════════════════
══════════
    DAY 8
═══════════════════════════════════════
══════════
Money: $156.40
Store: CLOSED

UPGRADES:
 ├── Manager: Level 3
 │  (Prices +30%)
 ├── Coffee Machine: Level 2
 │  (+4 customers)
 └── Decoration: Level 1
   (+1 customer)

INVENTORY:
Coffee: 12 units
Tea: 8 units
```

For Extended learners, Task 9 can use a "Manage Upgrades" submenu that dynamically lists available upgrades. The Shop could maintain a list of Upgrade objects, and the menu iterates through them displaying name, current level, next cost, and effect description.

Task 10's status display should call methods on Upgrade objects to retrieve their descriptions. Each upgrade is responsible for describing itself, e.g., "Manager Level 3 - Prices +30%" or "Coffee Machine Level 2 - Adds 4 daily customers". This separation of concerns minimizes changes required in the status display code when new upgrade types are added.

Session 3 Deliverables

Automated Operations

Complete daily processing with customer generation, transaction loops, and summarization

Analytics

Daily summaries tracking customers, sales, revenue, and conversion rates

Upgrade Systems

Three functional upgrade paths with cost formulas and visible effects

Enhanced Interface

Status screen showing upgrades, menu options for upgrades, daily summaries

Normal Path Checklist

- generate_all_customers() produces complete daily lists
- process_all_customers() loops through and handles transactions
- Daily summary displays customers, sales, revenue
- "Run business day" menu option works end-to-end
- Coffee machine, manager, decoration levels exist
- Upgrade functions implement exponential cost formula
- Upgrade functions check money and update state
- Customer count formula uses coffee machine and decoration
- Price formula uses manager level
- Menu has upgrade options
- Status screen shows all upgrade levels

Extended Path Checklist

- run_business_day() method orchestrates daily flow
- generate_customers_for_day() returns Customer objects
- process_customers() handles Customer objects
- Daily stats stored in shop's internal list
- Upgrade base class with cost calculation
- CoffeeMachineUpgrade, ManagerUpgrade, DecorationUpgrade subclasses
- Upgrades have apply_effect() or similar methods
- Shop stores list/dict of Upgrade objects
- Upgrade costs calculated via upgrade objects
- Status screen queries upgrade objects for display
- Clear separation between upgrade logic and shop logic

Playtesting Your Simulation

Before Session 4, playtest the simulation for 10-15 in-game days. Evaluate upgrade impact, financial balance, customer counts, and stock management. Playtesting identifies balance issues and bugs. Adjust formulas, costs, or item prices as needed.