Simple Delivery Service Program

Comprehensive Project Documentation

---

1. Cover Page

Project Title: Simple Delivery Service Program

Project Type: Command-Line Application (Educational Project)

Version: 1.0

Date: November 23, 2025

Prepared By: Development Team

Objective: To create a simple, menu-driven delivery service management system for order placement, status tracking, and delivery management.

---

2. Introduction

The Simple Delivery Service Program is a Python-based command-line application designed to manage delivery orders efficiently. It provides users with an intuitive interface to place new orders, update delivery statuses, check order information, and view all existing orders in real-time. The system demonstrates fundamental programming concepts including data structures, functions, user input handling, and control flow.

Target Users: - Delivery service managers - Order coordinators - Students learning Python programming

Project Scope: Educational demonstration of delivery order management system with in-memory data storage.

---

3. Problem Statement

Current Challenge: Manual delivery order management is time-consuming and prone to errors. There is a need for a simple, organized system to: - Track multiple delivery orders simultaneously - Maintain current status of each delivery - Provide quick access to order information - Allow status updates in real-time

Solution Proposed: A command-line application that centralizes delivery order management with easy-to-use functionality for all basic delivery operations.

---

4. Functional Requirements

Requirement ID	Requirement	Description
FR-1	Place Order	Users can create new delivery orders with customer name, item, and delivery address
FR-2	Update Status	Users can update delivery status (Preparing, Out for delivery, Delivered)
FR-3	Check Order	Users can retrieve and view specific order details using Order ID
FR-4	List All Orders	System displays summary of all orders with current statuses
FR-5	Generate Order ID	System automatically generates unique Order IDs for each order
FR-6	Menu Interface	System provides user-friendly menu for navigation
FR-7	Exit Program	Users can safely terminate the program

---

5. Non-Functional Requirements

Requirement ID	Requirement	Description
NFR-1	Usability	Simple, intuitive menu interface requiring minimal training
NFR-2	Performance	All operations should complete instantly
NFR-3	Data Validation	Basic validation for user inputs
NFR-4	Maintainability	Clean, well-commented code for easy modification
NFR-5	Reliability	Consistent behavior across multiple operations
NFR-6	Scalability	Can handle multiple orders during a session

---

6. System Architecture

6.1 High-Level Architecture

```
User Interface (CLI Menu)
        ↓
Input Processing Layer
        ↓
Business Logic Layer
  ├── Place Order
  ├── Update Status
  ├── Check Order
  └── Show All Orders
        ↓
Data Storage Layer (Dictionary)
        ↓
Output Display Layer
```

6.2 Architecture Description

Layer 1: User Interface - Command-line menu with numbered options - Text-based interaction prompts

Layer 2: Input Processing - Captures user choices and order details - Routes requests to appropriate functions

Layer 3: Business Logic - Core operations for order management - Data manipulation and status updates

Layer 4: Data Storage - In-memory dictionary for storing orders - Global counter for Order ID generation

Layer 5: Output Display - Formatted text output for users - Clear status and confirmation messages

---

7. Design Diagrams

7.1 Use Case Diagram

```
┌─────────────────────────────────────────────┐
│                     USER                    │
└────────────────┬────────────────────────────┘
                 │
    ┌────────────┼────────────┬──────────────┐
    │            │            │              │
    ▼            ▼            ▼              ▼
┌─────────┐  ┌──────────┐  ┌──────────┐  ┌────────────┐
│ Place   │  │ Update   │  │ Check    │  │ Show All   │
│ Order   │  │ Status   │  │ Order    │  │ Orders     │
└─────────┘  └──────────┘  └──────────┘  └────────────┘
    │            │            │              │
    └────────────┼────────────┴──────────────┘
                 │
        ┌────────▼─────────┐
        │  Delivery System │
        └──────────────────┘
```

7.2 Workflow Diagram

```
START
  ↓
Display Menu
  ↓
User Chooses Option
  ├─→ 1: Place Order → Get Input → Store in Dictionary → Generate Order ID
→ Display Confirmation
  ├─→ 2: Update Status → Input Order ID → Check Exists → Get New Status →
Update Dictionary
  ├─→ 3: Check Order → Input Order ID → Retrieve Details → Display Order
Information
  ├─→ 4: Show All Orders → Loop Through Dictionary → Display Summary
  ├─→ 5: Exit → Display Thank You Message → END
  └─→ Invalid → Display Error → Return to Menu
  ↓
Continue Loop or Exit
  ↓
END
```

7.3 Sequence Diagram

```
User          Menu               Order Function       Dictionary
 │              │                     │                  │
 ├─ Select 1 ──→│                     │                  │
 │              ├─ Call place_order() │                  │
 │              │                     ├─ Get Input       │
 │              │                     │←─ User Details   │
 │              │                     ├─ Store Order ───→│
 │              │                     │←─ Confirmation   │
 │              │←─ Display Message───│                  │
 │←─ Confirmation│                     │                  │
 │              │                     │                  │
 ├─ Select 2 ──→│                     │                  │
 │              ├─ Call update_status()                  │
 │              │                     ├─ Get Order ID    │
 │              │                     ├─ Lookup Order ──→│
 │              │                     │←─ Found          │
 │              │                     ├─ Update Status ─→│
 │              │←─ Display Message───│                  │
 │←─ Confirmation│                     │                  │
```

7.4 Class/Component Diagram

```
┌─────────────────────────────────────┐
│    Delivery Service System          │
├─────────────────────────────────────┤
│ Attributes:                         │
│  - deliveries: dict                 │
│  - order_id_counter: int            │
├─────────────────────────────────────┤
│ Methods:                            │
│  - main()                           │
│  - place_order()                    │
│  - update_status()                  │
│  - check_order()                    │
│  - show_all_orders()                │
└─────────────────────────────────────┘
         │
         ├─→ Order Dictionary
         │   ├─ Order ID (key)
         │   └─ Order Details (value)
         │       ├─ name
         │       ├─ item
         │       ├─ address
         │       └─ status
         │
         └─→ Global Variables
             ├─ deliveries {}
             └─ order_id_counter
```

7.5 ER Diagram (Conceptual Data Model)

```
┌───────────────────────────────┐
│          ORDER                │
├───────────────────────────────┤
│ Order_ID (PK)                 │
│ Customer_Name                 │
│ Item_to_Deliver               │
│ Delivery_Address              │
│ Status                        │
│ Timestamp (future feature)    │
└───────────────────────────────┘

Status Values:
├─ Preparing for delivery
├─ Out for delivery
└─ Delivered
```

---

8. Design Decisions & Rationale

8.1 Data Structure Choice: Dictionary

Decision: Use Python dictionary with Order ID as key.

Rationale: - O(1) lookup time for order retrieval - Simple to implement and understand - Flexible value structure (nested dictionary) - Suitable for in-memory storage

8.2 Global Variables

Decision: Use global variables for `deliveries` and `order_id_counter`.

Rationale: - Simplicity for educational purpose - Easy to understand program flow - Maintains state across function calls - Note: Not recommended for production code

8.3 User Interface: Command-Line Menu

Decision: Use text-based menu interface instead of GUI.

Rationale: - Easier to implement and understand - Cross-platform compatibility - Suitable for learning and demonstrations - Minimal dependencies

8.4 Status Management

Decision: Use predefined status options (1, 2, 3).

Rationale: - Prevents invalid status entries - Ensures data consistency - Simple user interaction - Easy to extend in future

8.5 Order ID Generation

Decision: Auto-incrementing integer starting from 1.

Rationale: - Simple implementation - Guarantees uniqueness - Easy to remember and communicate - Sequential tracking capability

---

9. Implementation Details

9.1 Data Structure

```
deliveries = {
    1: {
        "name": "John Doe",
        "item": "Electronics",
        "address": "123 Main St",
        "status": "Preparing for delivery"
    },
    2: {
        "name": "Jane Smith",
        "item": "Books",
        "address": "456 Oak Ave",
        "status": "Out for delivery"
    }
}
```

9.2 Function Descriptions

place_order(): - Captures customer name, item, delivery address - Stores order in dictionary with auto-generated ID - Displays confirmation with Order ID

update_status(): - Prompts for Order ID - Checks if order exists - Displays status options (1, 2, 3) - Updates dictionary with new status

check_order(): - Requests Order ID from user - Retrieves order details from dictionary - Displays formatted order information - Handles order not found scenario

show_all_orders(): - Iterates through dictionary - Displays summary line for each order - Handles empty deliveries scenario

main(): - Infinite loop for menu display - Routes user choice to appropriate function - Handles invalid inputs

9.3 Control Flow

```
main() loop
├─ Display Menu
├─ Get User Choice
├─ If Choice == 1 → place_order()
├─ If Choice == 2 → update_status()
├─ If Choice == 3 → check_order()
├─ If Choice == 4 → show_all_orders()
├─ If Choice == 5 → Exit Loop
└─ Else → Display Error
```

---

10. Screenshots / Results

10.1 Main Menu Display

```
====== Delivery Service Menu ======
1. Place New Order
2. Update Delivery Status
3. Check Order Status
4. Show All Orders
5. Exit

Choose an option:
```

10.2 Place Order Example

```
--- Place a New Delivery Order ---
Customer Name: John Doe
Item to Deliver: Laptop
Delivery Address: 123 Main Street

Order placed successfully! Your Order ID is: 1
```

10.3 Check Order Status Example

```
--- Check Delivery Status ---
Enter Order ID: 1

Order ID: 1
Customer: John Doe
Item: Laptop
Address: 123 Main Street
Status: Out for delivery
```

10.4 Show All Orders Example

```
--- All Delivery Orders ---
Order 1: Laptop for John Doe - Out for delivery
Order 2: Books for Jane Smith - Preparing for delivery
Order 3: Package for Mike Johnson - Delivered
```

---

11. Testing Approach

11.1 Unit Testing Strategy

Test Case	Input	Expected Output	Status
TC-1	Place order with valid details	Order ID generated and displayed	Pass
TC-2	Place multiple orders	Each receives unique ID	Pass
TC-3	Update status with valid ID	Status updated in dictionary	Pass
TC-4	Update status with invalid ID	“Order ID not found” message	Pass
TC-5	Check order with valid ID	All order details displayed	Pass
TC-6	Check order with invalid ID	“Order not found” message	Pass
TC-7	Show orders when empty	“No orders yet” message	Pass
TC-8	Invalid menu choice	“Invalid option” message	Pass
TC-9	Exit program	Confirmation message and termination	Pass

11.2 User Acceptance Testing

Scenario 1: Complete Order Lifecycle 1. Place an order 2. View the order 3. Update status to “Out for delivery” 4. Update status to “Delivered” 5. Verify final status

Scenario 2: Multiple Orders 1. Place 3 different orders 2. Show all orders 3. Update one order’s status 4. Verify others remain unchanged

Scenario 3: Error Handling 1. Try to update non-existent order 2. Try to check non-existent order 3. Enter invalid menu choice 4. Verify error messages displayed

---

12. Challenges Faced

12.1 Challenge 1: Global Variables Management

Issue: Managing global state across functions can lead to confusion. **Solution:** Used global keyword explicitly and documented global variable usage.

12.2 Challenge 2: Input Validation

Issue: Basic input validation could fail on unexpected inputs. **Solution:** Added try-except blocks for integer conversion in order ID input.

12.3 Challenge 3: Data Persistence

Issue: All data is lost when program exits. **Solution:** Documented this limitation and suggested file/database storage as future enhancement.

12.4 Challenge 4: Scalability

Issue: Infinite dictionary growth could cause memory issues in long-running session. **Solution:** Not a concern for educational use; recommended implementing data cleanup in production.

12.5 Challenge 5: User Experience

Issue: Text-based interface lacks visual appeal. **Solution:** Used clear formatting, status messages, and organized menu structure.

---

13. Learnings & Key Takeaways

13.1 Programming Concepts Applied

- **Data Structures:** Dictionary usage for efficient data storage and retrieval
- **Functions:** Modular design with single responsibility principle
- **Control Flow:** Menu-driven loops and conditional logic
- **Global Variables:** State management (educational demonstration)
- **User Input:** Input validation and error handling

13.2 Software Engineering Principles

- **Modularity:** Each function handles one specific task
- **Readability:** Clear variable names and code comments
- **Error Handling:** Validation and user-friendly error messages
- **User Interface Design:** Simple, intuitive menu structure

13.3 Best Practices Learned

- Use meaningful function and variable names
- Add comments for clarity (especially for beginners)
- Provide clear feedback to users after operations
- Validate input before processing
- Design for extension and modification

13.4 Improvements from Refactoring

- Added comments and docstrings
- Better error messages
- Consistent formatting throughout
- Separated concerns into distinct functions

---

14. Future Enhancements

14.1 Phase 2 Features

- **Persistent Storage:** Save orders to file or database
- **User Authentication:** Multi-user support with login
- **Advanced Filtering:** Search orders by customer, item, or status
- **Delivery Time Tracking:** Add timestamp for order creation and delivery
- **Route Optimization:** Calculate optimal delivery routes

14.2 Phase 3 Features

- **Web Interface:** Convert to web application (Flask/Django)

- **Mobile App:** Native mobile application for delivery personnel
- **Real-time Notifications:** SMS/Email alerts for status updates
- **Payment Integration:** Online payment processing
- **Analytics Dashboard:** Delivery metrics and performance tracking

14.3 Technical Improvements

- Implement OOP with Order class
- Add unit testing framework (pytest)
- Implement database (SQLite, PostgreSQL)
- Add API endpoints (REST/GraphQL)
- Containerization (Docker)
- Continuous Integration/Deployment

14.4 User Experience Enhancements

- GUI with tkinter or PyQt
- Automated email confirmations
- QR code generation for orders
- Receipt printing functionality
- Multi-language support

---

15. References

15.1 Python Documentation

- Python Official Documentation: https://docs.python.org/3/
- Python Data Structures (Dictionaries): https://docs.python.org/3/tutorial/datastructures.html
- Python Functions: https://docs.python.org/3/tutorial/controlflow.html

15.2 Software Engineering Resources

- Software Design Documents Best Practices: https://www.atlassian.com/work-management/knowledge-sharing/documentation/software-design-document
- Python Project Documentation: https://realpython.com/documenting-python-code/
- Clean Code Principles: https://www.python.org/dev/peps/pep-0008/

15.3 Related Technologies

- Flask Web Framework: https://flask.palletsprojects.com/
- SQLite Database: https://www.sqlite.org/
- Git Version Control: https://git-scm.com/

15.4 Learning Resources

- Real Python Tutorials: https://realpython.com/
- W3Schools Python: https://www.w3schools.com/python/
- GeeksforGeeks Python: https://www.geeksforgeeks.org/python-tutorial/

Appendix

A. Complete Code Structure

The program consists of: - 1 main module file - 5 core functions - 1 main loop - Global data storage

B. Installation Instructions

1. Ensure Python 3.x is installed
2. Save the code as `delivery_service.py`
3. Run: `python delivery_service.py`
4. No additional packages required

C. Glossary

Term	Definition
Order ID	Unique identifier for each delivery order
Status	Current state of delivery (Preparing, Out for delivery, Delivered)
In-memory	Data stored in RAM, lost when program exits
Dictionary	Python data structure for key-value pairs
Global Variable	Variable accessible throughout the program
CLI	Command-Line Interface
Function	Reusable block of code performing a specific task

Document Version: 1.0
Last Updated: November 23, 2025
Status: Complete
Approval: Ready for Distribution

End of Document