

โครงงาน Python File I/O

เรื่อง ระบบขายรถยนต์มือ 2 JB Garage

จัดทำโดย

| | | |
|--------------|-----------|---------------|
| นายภูริณัฐ | แสนยางนอก | 6706022610209 |
| นายอภิชา | ภาอ่อน | 6706022610454 |
| นายวุฒิชัย | กล้าชุ่ม | 6706022610357 |
| นายสุธินันท์ | ครองแถว | 6706022610349 |

โครงงานนี้เป็นส่วนหนึ่งของการศึกษา หลักสูตรวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมสารสนเทศและเครือข่าย ภาควิชาเทคโนโลยีสารสนเทศ
คณะเทคโนโลยีและการจัดการและอุตสาหกรรม มหาวิทยาลัยเทคโนโลยีพระ
จอมเกล้าพระนครเหนือ

ภาคเรียนที่ 1 ปีการศึกษา 2568

ลิขสิทธิ์ของมหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

คำนำ

โครงการ “ระบบขายรถยนต์มือสอง JB Garage” จัดทำขึ้นเพื่อเป็นส่วนหนึ่งของรายวิชา Python File I/O โดยมีวัตถุประสงค์เพื่อศึกษาและประยุกต์ใช้ความรู้ด้านการเขียนโปรแกรมภาษา Python ในการจัดการข้อมูลด้วยไฟล์ (File Input/Output) โดยสร้างระบบที่สามารถบันทึก แก้ไข ค้นหา และแสดงผลข้อมูลรถยนต์มือสองได้อย่างมีประสิทธิภาพ

การจัดทำโครงการนี้ช่วยให้ผู้จัดทำได้เรียนรู้กระบวนการพัฒนาโปรแกรมเชิงปฏิบัติ ตั้งแต่การออกแบบระบบ การเขียนโค้ด ไปจนถึงการทดสอบการทำงาน อีกทั้งยังช่วยฝึกทักษะการทำงานเป็นทีม การแก้ปัญหา และการนำความรู้ไปประยุกต์ใช้กับงานในชีวิตจริงได้

คณะผู้จัดทำหวังเป็นอย่างยิ่งว่าโครงการนี้จะเป็นประโยชน์ต่อผู้ที่สนใจศึกษาและพัฒนา ระบบจัดการข้อมูลด้วยภาษา Python และสามารถนำแนวคิดไปต่อยอดในอนาคตได้

สุดท้ายนี้ คณะผู้จัดทำขอขอบพระคุณอาจารย์ผู้สอนที่ได้ให้ความรู้และคำแนะนำตลอดจน การสนับสนุนในการจัดทำโครงการนี้จนสำเร็จลุล่วงไปได้ด้วยดี

คณะผู้จัดทำ

นายภูริณัฐ แสนยางนอก

นายอภิชา เกาอ่อน

นายวุฒิชัย กล้าชุ่ม

นายสุธินันท์ ครองแถว

สารบัญ

| เรื่อง | หน้า |
|---|------|
| คำนำ | ก |
| สารบัญ | ข |
| สารบัญภาพ | ค |
| สารบัญตาราง | ง |
| บทที่ 1 บทนำ | 1 |
| 1.1 วัตถุประสงค์ของโครงการ | 2 |
| 1.2 ขอบเขตของโครงการ | 3 |
| 1.3 ประโยชน์ที่ได้รับ | 4 |
| 1.4 เครื่องมือที่คาดว่าจะต้องใช้ | 6 |
| บทที่ 2 Function Main | 8 |
| 2.1 ส่วนที่ 1: การเรียกใช้โมดูลฟังก์ชันอื่น | 8 |
| 2.2 ส่วนที่ 2: การกำหนดฟังก์ชันหลัก | 8 |
| 2.3 ส่วนที่ 3: การวนลูปแสดงเมนูหลัก | 8 |
| 2.4 ส่วนที่ 6: การใช้ match-case เพื่อเลือกเมนูที่ต้องการ | 9 |
| 2.5 ส่วนที่ 7: เมนูย่อยสำหรับ “ดูข้อมูลรถยนต์” | 10 |
| 2.6 ส่วนสุดท้าย จุดเริ่มต้นของโปรแกรม | 10 |
| บทที่ 3 การจัดการข้อมูลและการนำเข้าโมดูล (Import & Data Management) | 12 |
| 3.1 การนำเข้าโมดูล (Import Modules) | 12 |
| 3.2 การกำหนดชื่อไฟล์เก็บข้อมูล | 13 |
| 3.3 การกำหนดโครงสร้างข้อมูลด้วย Struct | 13 |
| 3.4 ฟังก์ชันช่วยเหลือในการจัดการข้อความ | 15 |
| 3.5 การบันทึกข้อมูลทั้งหมด (Save All) | 16 |
| 3.6 การโหลดข้อมูลทั้งหมด (Load All) | 21 |
| 3.7 การเปิดไฟล์ | 21 |
| 3.8 สรุปภาพรวมของการจัดการข้อมูล | 26 |

| | |
|---|----|
| บทที่ 4 การอธิบายการทำงานของ Function Add | 27 |
| บทที่ 5 Function Delete | 32 |
| บทที่ 6 Function View () | 33 |
| บทที่ 7 การตกแต่ง | 39 |
| 7.1 จุดเริ่มต้น: ปัญหาที่เราต้องแก้ | 39 |
| 7.2 แปลงไฟล์เป็นข้อมูลที่โปรแกรมเข้าใจ | 39 |
| 7.3 แยกกรดตามสถานะ | 39 |
| 7.4 ออกแบบตารางให้อ่านง่าย | 39 |
| 7.5 สรุปข้อมูล (Summary) | 40 |
| 7.6 ความซับซ้อนของงาน | 40 |
| บทที่ 8 Summary และ การไหลของข้อมูล | 41 |
| 8.1 การไหลของข้อมูลในรายงาน | 41 |
| 4.6.1 ฟังก์ชัน make_table_not_sold() | 43 |
| 4.6.2 ฟังก์ชัน make_table_sold() | 44 |
| 4.6.3 ฟังก์ชัน make_summary() | 45 |
| บทที่ 9 คู่มือการใช้งานระบบซื้อ – ขายรถยนต์มือสอง | 46 |
| 3.1 เมนูหลักของโปรแกรมการซื้อขายรถมือสอง | 46 |
| 3.2 เมนูย่อยการแสดงผลข้อมูล (View Car) | 46 |
| 3.3 เมนู Single car | 47 |
| 3.4 เมนู All car | 48 |
| 3.5 เมนู Filter | 48 |
| 3.6 เมนู Car not sale + Summary | 49 |
| 3.7 เมนู Sold Car | 50 |
| 3.8 เมนูการเพิ่มข้อมูล (Add Car) | 51 |
| 3.8 เมนูการเพิ่มข้อมูล (Add Car) | 52 |
| 3.8 การลบข้อมูล (Delete Car) | 52 |
| บทที่ 10 สรุป อภิปรายผล และข้อเสนอแนะ | 54 |

| | | |
|----------|---|----|
| 10.1 | สรุปผลการพัฒนาโปรแกรม | 54 |
| 10.2 | 5.2 อภิปรายผลการทำงานของโปรแกรม | 54 |
| 10.3 | 5.4 ข้อเสนอแนะในการพัฒนาในอนาคต | 56 |
| 10.4 | Flow Diagram การทำงานของโปรแกรม (Workflow) | 57 |
| 10.5 | สรุปภาพรวม | 58 |
| บทที่ 11 | สิ่งที่คิดว่า Programs นี้ควรได้รับการแก้ไข | 59 |
| | Prototype / จุดสังเกตเพื่อพัฒนาต่อในอนาคต (Future Improvements Prototype) | 59 |
| 11.1 | 1. การตรวจสอบและความถูกต้องของข้อมูล (Input Validation & Data Integrity) | 59 |
| 11.2 | การจัดเก็บข้อมูลและความปลอดภัย (File Handling & Storage Safety) | 60 |
| 11.3 | ประสบการณ์ผู้ใช้ (User Experience & UI) | 60 |
| 11.4 | รายงานและสถิติ (Reporting & Statistics) | 61 |
| 11.5 | ประสิทธิภาพและขยายตัว (Performance & Scalability) | 61 |
| 11.6 | ความปลอดภัยและระบบตรวจสอบ (Security & Logging) | 62 |
| 11.7 | ฟังก์ชันเสริมที่แนะนำในอนาคต | 62 |
| 11.8 | สรุป Prototype: | 63 |

สารบัญภาพ

| เรื่อง | หน้า |
|--|------|
| ภาพที่ 2-1 ภาพการ Import Function | 8 |
| ภาพที่ 2-2 Function Main () | 8 |
| ภาพที่ 2-3 การวนลูปแสดงเมนูหลัก | 8 |
| ภาพที่ 2-4 การรับ input จากผู้ใช้ และ การตรวจสอบ ValueError | 9 |
| ภาพที่ 2-5 การใช้ Match – Case | 9 |
| ภาพที่ 2-6 Choice ในการดูข้อมูลรถยนต์ | 10 |
| ภาพที่ 2-7 การเรียกใช้ Main() | 10 |
| ภาพที่ 3-1 Module ที่ import | 12 |
| ภาพที่ 3-2 ชื่อไฟล์ที่จัดเก็บข้อมูล ทั้ง 3 ไฟล์ | 13 |
| ภาพที่ 3-3 Struct_basic | 14 |
| ภาพที่ 3-4 Struct_Status | 14 |
| ภาพที่ 3-5 โค้ดที่ช่วยในการจัดการตาราง และการ Encode | 15 |
| ภาพที่ 3-6 การเปิดไฟล์ | 16 |
| ภาพที่ 3-7 Loop Cars Information | 16 |
| ภาพที่ 3-8 บันทึกไฟล์ข้อมูลพื้นฐาน basic.dat | 17 |
| ภาพที่ 3-9 บันทึกสถานะรถ | 17 |
| ภาพที่ 3-10 บันทึกข้อมูลการขายจริง | 18 |
| ภาพที่ 3-11 ตารางเก็บข้อมูลที่ Report อิงจาก Not Sale.txt | 20 |
| ภาพที่ 3-12 การตรวจสอบ File | 21 |
| ภาพที่ 3-13 การเปิดไฟล์ | 21 |
| ภาพที่ 3-14 การวนค่าที่ละ Block | 22 |
| ภาพที่ 3-15 การ Unpack ข้อมูล | 22 |
| ภาพที่ 3-16 การรวมข้อมูลทั้งหมด เพื่อนำไปใช้ต่อ | 23 |
| ภาพที่ 4-1 โหลดรถทั้งหมดเป็น list ของ dict เพื่อใช้ตรวจสอบ CarID ซ้ำ | 27 |
| ภาพที่ 4-2 สร้าง Dictionary รถใหม่ | 27 |
| ภาพที่ 4-3 เพิ่มข้อมูลลงใน List ของรถและบันทึก | 28 |

| | |
|--|----|
| ภาพที่ 4-4 ฟังก์ชัน Update() | 28 |
| ภาพที่ 4-5 กรณีรถยังไม่ได้ขาย | 29 |
| ภาพที่ 4-6 กรณีรถขายแล้ว อัปเดตข้อมูล | 30 |
| ภาพที่ 4-7 ส่วนสรุปและบันทึกไฟล์ | 31 |
| ภาพที่ 5-1 ฟังก์ชัน Delete() | 32 |
| ภาพที่ 6-1 View(1) – แสดงรถเพียงคันเดียว | 33 |
| ภาพที่ 6-2 View(2) แสดงรถทั้งหมด | 34 |
| ภาพที่ 6-3 View(3) กรองข้อมูล (Filter) | 34 |
| ภาพที่ 6-4 Filter brands | 35 |
| ภาพที่ 6-5 Filter Models | 35 |
| ภาพที่ 6-6 Filter yeasers | 35 |
| ภาพที่ 6-7 Filter status | 36 |
| ภาพที่ 6-8 แสดงผลตาราง เหมือน All Cars | 36 |
| ภาพที่ 6-9 เมนู View(4) | 37 |
| ภาพที่ 6-10 เมนู View(5) | 37 |
| ภาพที่ 7-1 หน้าตา Summary ที่คาดหวัง | 40 |
| ภาพที่ 8-1 โค้ด make_table_not_sold() | 43 |
| ภาพที่ 8-2 ฟังก์ชัน make_table_sold() | 44 |
| ภาพที่ 8-3 ฟังก์ชัน make_summary() | 45 |
| ภาพที่ 9-1 แสดงเมนูหลักของระบบ | 46 |
| ภาพที่ 9-2 เมนู view | 47 |
| ภาพที่ 9-3 เมนู single car | 47 |
| ภาพที่ 9-4 เมนู All car | 48 |
| ภาพที่ 9-5 ผลการแสดงผล All car | 48 |
| ภาพที่ 9-6 การแสดงผล fliter | 49 |
| ภาพที่ 9-7 ภาพการใช้งาน fliter | 49 |
| ภาพที่ 9-8 การใช้งาน | 50 |
| ภาพที่ 9-9 การแสดงผลรัน | 50 |
| ภาพที่ 9-10 การใช้งาน | 51 |

| | |
|----------------------------------|----|
| ภาพที่ 9-11 ภาพการแสดงผลการทำงาน | 51 |
| ภาพที่ 9-12 การใช้งาน Add | 51 |
| ภาพที่ 9-13 การใช้งาน update | 52 |
| ภาพที่ 9-14 การใช้งาน delete | 53 |

สารบัญตาราง

| เรื่อง | หน้า |
|---|------|
| ตารางที่ 1 Library ที่ใช้ใน Programs | 6 |
| ตารางที่ 2 สรุปการทำงานของฟังก์ชัน Main | 11 |
| ตารางที่ 3 ข้อมูลที่แต่ละ File จัดเก็บ | 13 |
| ตารางที่ 4 ข้อมูลในตาราง Struct_Basic | 14 |
| ตารางที่ 5 ข้อมูลที่จัดเก็บใน Struct_Status | 15 |
| ตารางที่ 6 ข้อมูลจริงใน File | 20 |
| ตารางที่ 7 ข้อมูลที่เก็บเป็น Binary | 26 |
| ตารางที่ 8 โครงสร้างการเก็บข้อมูล | 54 |
| ตารางที่ 9 ตารางสรุปสถิติ (Statistics Summary) | 56 |
| ตารางที่ 10 ตาราง Check List ข้อมูลที่ควรปรับแก้ | 59 |
| ตารางที่ 11 ตารางข้อมูลการ Back Up | 60 |
| ตารางที่ 12 ตารางเปรียบเทียบประสบการณ์ผู้ใช้ | 61 |
| ตารางที่ 13 การจัดการรายงาน และ สถิติ | 61 |
| ตารางที่ 14 ประสิทธิภาพที่รองรับข้อมูลในตอนนี้ | 62 |
| ตารางที่ 15 การเปรียบเทียบเรื่องความปลอดภัยของระบบ และ การตรวจสอบ | 62 |

บทที่ 1

บทนำ

รายงานฉบับนี้จัดทำขึ้นเพื่อเป็นส่วนหนึ่งของการศึกษาในรายวิชา Computer Programming โดยมีจุดมุ่งหมายในการพัฒนาโปรแกรม JB Garage Used Car System ซึ่งเป็นระบบสำหรับจัดเก็บและบริหารจัดการข้อมูลรถยนต์มือสองอย่างเป็นระบบระเบียบและมีประสิทธิภาพ

โปรแกรมที่พัฒนาขึ้นสามารถดำเนินการได้ทั้งในด้านการเพิ่ม แก้ไข ลบ และค้นหาข้อมูลรถยนต์ รวมถึงการตรวจสอบสถานะการขายและสรุปผลข้อมูลสำคัญต่าง ๆ เพื่อรองรับการจัดการข้อมูลที่ซับซ้อน โดยโครงสร้างของระบบอาศัยแฟ้มข้อมูล 3 ส่วน ได้แก่ cars_basic.dat ซึ่งเก็บข้อมูลพื้นฐานของรถ เช่น รหัสรถ ปีที่ผลิต ยี่ห้อ รุ่น เลขไมล์ และราคาซื้อ, cars_status.dat สำหรับเก็บสถานะของรถ ได้แก่ การใช้งาน การขาย และราคาขาย, และ cars_sale.dat ซึ่งเก็บรายละเอียดการขาย เช่น ชื่อลูกค้า เบอร์ติดต่อ และวันที่ซื้อรถ ทั้งนี้เพื่อให้ระบบสามารถทำงานได้อย่างครบถ้วนและเชื่อมโยงกัน

การจัดทำรายงานและโปรแกรมนี้นี้ทำให้ผู้จัดทำมีโอกาสฝึกทักษะการเขียนโปรแกรมเชิงโครงสร้าง การออกแบบระบบจัดการข้อมูล และการวิเคราะห์ปัญหาเชิงระบบ ซึ่งเป็นพื้นฐานสำคัญในการเรียนรู้และพัฒนาต่อยอดในด้านเทคโนโลยีสารสนเทศและวิทยาการคอมพิวเตอร์

คณะผู้จัดทำขอกราบขอบพระคุณอาจารย์ผู้สอนที่ได้ให้คำแนะนำและมอบหมายงานอันทรงคุณค่า ทำให้ผู้จัดทำได้รับความรู้และประสบการณ์ที่เป็นประโยชน์อย่างยิ่ง คณะผู้จัดทำหวังเป็นอย่างยิ่งว่า รายงานฉบับนี้จะบรรลุตามวัตถุประสงค์และเป็นประโยชน์ต่อผู้ที่สนใจ หากมีข้อผิดพลาดประการใด คณะผู้จัดทำขอน้อมรับไว้เพื่อปรับปรุงแก้ไขในโอกาสต่อไป

ลงชื่อ

| | | |
|--------------|-----------|---------------|
| นายภูริณัฐ | แสนยางนอก | 6706022610209 |
| นายอภิชา | เกาอ่อน | 6706022610454 |
| นายจุฒิชัย | กล้าชุม | 6706022610357 |
| นายสุธินันท์ | ครองแถว | 6706022610349 |

(ชื่อผู้จัดทำ)

1.1 วัตถุประสงค์ของโครงการ

โครงการนี้มีวัตถุประสงค์หลักในการพัฒนาโปรแกรม JB Garage Used Car System เพื่อให้สามารถจัดเก็บและบริหารจัดการข้อมูลรถยนต์มือสองได้อย่างมีประสิทธิภาพ โดยมีรายละเอียดดังนี้

- 1.1.1 เพื่อพัฒนาโปรแกรมที่สามารถ **เพิ่ม แก้ไข ลบ และค้นหาข้อมูลรถยนต์** ได้อย่างสะดวกและเป็นระบบ
- 1.1.2 เพื่อออกแบบระบบที่สามารถ **จัดเก็บข้อมูลรถยนต์ในรูปแบบแฟ้มข้อมูล (File-based System)** ได้แก่ cars_basic.dat, cars_status.dat และ cars_sale.dat ซึ่งเชื่อมโยงกันด้วยรหัสรถ (car_id)
- 1.1.3 เพื่อให้สามารถ **ตรวจสอบสถานะรถยนต์** ได้ เช่น รถที่ขายแล้ว รถที่ยังไม่ขาย รวมถึงการสรุปข้อมูลการขายได้อย่างชัดเจน
- 1.1.4 เพื่อเก็บข้อมูลลูกค้าและการซื้อขายที่เกี่ยวข้องกับรถยนต์แต่ละคันอย่างครบถ้วน เช่น ชื่อลูกค้า เบอร์ติดต่อ และวันที่ซื้อขาย
- 1.1.5 เพื่อเสริมสร้างทักษะของผู้จัดทำในการ **เขียนโปรแกรมเชิงโครงสร้าง การจัดการข้อมูล และการออกแบบระบบเชิงปฏิบัติ** ซึ่งสามารถนำไปประยุกต์ใช้ในงานจริงได้

1.2 ขอบเขตของโครงการ

โครงการ JB Garage Used Car System มีการกำหนดขอบเขตการทำงานของโปรแกรมไว้ดังนี้

1.2.1 โปรแกรมสามารถทำงานในรูปแบบ Command Line Interface (CLI) โดยผู้ใช้งานสามารถสั่งงานผ่านเมนูที่กำหนดไว้

1.2.2 สามารถ เพิ่ม แก้ไข ลบ และค้นหาข้อมูลรถยนต์ ได้ โดยข้อมูลพื้นฐานที่จัดเก็บได้แก่

- รหัสรถ (Car ID)
- ปีที่ผลิต (Year)
- ยี่ห้อ (Brand)
- รุ่น (Model)
- เลขไมล์ (Odometer)
- ราคาซื้อ (Buy Price)

1.2.3 โปรแกรมสามารถเก็บข้อมูล สถานะของรถ ได้แก่ รถที่ยังไม่ได้ขาย รถที่ขายแล้ว สถานะการใช้งาน และราคาขาย

1.2.4 โปรแกรมสามารถเก็บข้อมูล การขายรถ ได้แก่ ชื่อลูกค้า เบอร์ติดต่อ และวันที่ซื้อขาย เพื่อเชื่อมโยงกับรถแต่ละคัน

1.2.5 โปรแกรมรองรับการ เรียกดูข้อมูล ได้หลายรูปแบบ เช่น

- แสดงรถเพียงคันเดียว
- แสดงรถทั้งหมด
- แสดงผลการกรองข้อมูล (Filter)
- สรุปรถที่ยังไม่ขาย
- สรุปรถที่ขายแล้วพร้อมรายละเอียดลูกค้า

1.2.6 ข้อมูลทั้งหมดจะถูกจัดเก็บไว้ในแฟ้มข้อมูล 3 แฟ้ม

ได้แก่ cars_basic.dat, cars_status.dat, และ cars_sale.dat โดยเชื่อมโยงกันด้วย car_id

1.3 ประโยชน์ที่ได้รับ

การพัฒนาโปรแกรม JB Garage Used Car System ส่งผลให้เกิดประโยชน์ในหลายด้าน ทั้งในเชิงการเรียนรู้และการประยุกต์ใช้งานจริง ดังนี้

1.3.1 ด้านผู้พัฒนา (นักศึกษา)

- ได้ฝึกทักษะการเขียนโปรแกรมเชิงโครงสร้างและการประยุกต์ใช้ภาษา Python ในการพัฒนาระบบจริง
- เรียนรู้การออกแบบโครงสร้างข้อมูล การใช้แฟ้มข้อมูล (File Handling) และการจัดการข้อมูลแบบมีระบบ
- เสริมสร้างทักษะการคิดเชิงวิเคราะห์ (Analytical Thinking) และการแก้ไขปัญหา (Problem Solving)
- เพิ่มประสบการณ์ในการทำงานเป็นทีม การแบ่งหน้าที่รับผิดชอบ และการสื่อสารเชิงวิชาการผ่านการทำรายงาน

1.3.2 ด้านผู้ใช้งาน (ร้านรถมือสองหรือผู้สนใจทั่วไป)

- สามารถจัดเก็บข้อมูลรถยนต์มือสองได้อย่างเป็นระบบ เช่น รหัสรถ ยี่ห้อ รุ่น ปีที่ผลิต เลขไมล์ ราคา และสถานการณ์ขาย
- ลดความซ้ำซ้อนและความผิดพลาดในการบันทึกข้อมูลด้วยระบบที่มีฟังก์ชันเพิ่ม แก้ไข ลบ และค้นหาข้อมูลได้อย่างสะดวก
- ช่วยให้ผู้ประกอบการสามารถตรวจสอบสถานะรถยนต์และติดตามข้อมูลการขายได้อย่างรวดเร็ว
- รองรับการวิเคราะห์เบื้องต้น เช่น การดูแนวโน้มการขายรถ หรือการประเมินข้อมูลรถที่ได้รับความนิยม

1.3.3 ด้านวิชาการและการต่อยอด

- รายงานและโครงงานนี้เป็นกรณีศึกษาเชิงปฏิบัติที่สามารถนำไปใช้เป็นต้นแบบในการพัฒนาระบบจัดการข้อมูลอื่น ๆ ได้
- สามารถต่อยอดสู่การเชื่อมต่อกับฐานข้อมูลขนาดใหญ่ (Database) หรือการพัฒนาเป็นระบบเว็บแอปพลิเคชันในอนาคต

- ช่วยให้นักศึกษาได้เตรียมความพร้อมสำหรับการทำงานจริงในสาขาเทคโนโลยีสารสนเทศ และวิทยาการคอมพิวเตอร์

1.4 เครื่องมือที่คาดว่าจะต้องใช้

ในการพัฒนาโครงการเรื่อง “ระบบขายรถยนต์มือสอง JB Garage” ผู้จัดทำได้ใช้เครื่องมือและโปรแกรมต่าง ๆ เพื่อให้การพัฒนากระบวนการเป็นไปอย่างมีประสิทธิภาพ ดังนี้

1.4.1 ภาษาโปรแกรม Python

ใช้ภาษา Python (เวอร์ชัน 3.10 ขึ้นไป) เนื่องจากเป็นภาษาที่มีโครงสร้างเข้าใจง่าย เหมาะสำหรับผู้เริ่มต้นและงานพัฒนาระบบขนาดเล็กถึงขนาดกลาง อีกทั้งยังมีไลบรารีมาตรฐานจำนวนมากที่สามารถนำมาใช้งานได้โดยไม่ต้องติดตั้งเพิ่มเติม เช่น การจัดการไฟล์ การแปลงข้อมูล และการตรวจสอบรูปแบบข้อมูล

1.4.2 โปรแกรม Visual Studio Code (VS Code)

เป็นโปรแกรม Integrated Development Environment (IDE) ที่ใช้สำหรับเขียนและรันโปรแกรมภาษา Python มีความสามารถในการตรวจสอบโค้ด แสดงสีไวยากรณ์ (Syntax Highlighting) และดีบั๊ก (Debug) ได้สะดวก อีกทั้งยังมีส่วนขยาย (Extension) สำหรับ Python ที่ช่วยให้การเขียนโปรแกรมง่ายและรวดเร็วยิ่งขึ้น

1.4.3 ไลบรารีที่ใช้ในโปรแกรม

โปรแกรมนี้อาศัยไลบรารีมาตรฐานของ Python เพื่อช่วยในการทำงาน ดังนี้

| ไลบรารี | หน้าที่ |
|---------|---|
| os | ใช้ตรวจสอบและจัดการไฟล์ภายในระบบ เช่น ตรวจสอบว่าไฟล์มีอยู่หรือไม่ |
| struct | ใช้สำหรับแปลงข้อมูลให้อยู่ในรูปแบบไบนารี เพื่อจัดเก็บลงไฟล์ .dat อย่างมีประสิทธิภาพ |
| re | ใช้สำหรับตรวจสอบรูปแบบของข้อมูล เช่น การตรวจสอบรหัสให้อยู่ในรูปแบบที่ถูกต้อง |

ตารางที่ 1 Library ที่ใช้ใน Programs

1.4.4 เครื่องมือจัดเก็บและจัดการข้อมูล

ใช้ไฟล์นามสกุล **.dat** เพื่อเก็บข้อมูลหลักของระบบ เช่น ข้อมูลรถยนต์ ข้อมูลสถานะ และข้อมูลการขาย โดยใช้รูปแบบการจัดเก็บแบบไบนารี (Binary File) เพื่อให้โปรแกรมสามารถอ่านและเขียนข้อมูลได้รวดเร็วและปลอดภัยจากการแก้ไขโดยตรง

1.4.5 เครื่องมืออื่น ๆ ที่เกี่ยวข้อง

- **Notepad++** : ใช้เปิดดูไฟล์ข้อความ เช่น รายงานผลการขายที่ระบบสร้างขึ้นในรูปแบบ **.txt**
- **Git / GitHub (ถ้ามี)** : ใช้ในการเก็บและจัดการเวอร์ชันของโค้ดในระหว่างการพัฒนา
- **Draw.io หรือ Microsoft Word** : ใช้ในการออกแบบแผนภาพระบบและจัดทำรายงานประกอบโครงการ

1.4.6 ระบบปฏิบัติการที่ใช้ในการพัฒนา

ครงงานนี้ถูกพัฒนาบนระบบปฏิบัติการ **Windows 10 / macOS** ซึ่งรองรับการติดตั้ง Python และการทำงานของโปรแกรม Visual Studio Code ได้อย่างสมบูรณ์

สรุป


เครื่องมือทั้งหมดที่กล่าวมามีบทบาทสำคัญในการพัฒนาและทดสอบระบบขายรถยนต์มือสอง JB Garage ช่วยให้ผู้ใช้จัดทำสามารถสร้างระบบที่มีโครงสร้างชัดเจน ใช้งานได้จริง และสามารถบันทึกข้อมูลได้อย่างมีประสิทธิภาพ

บทที่ 2

Function Main

ฟังก์ชัน `main()` เป็นฟังก์ชันหลักของโปรแกรม ซึ่งทำหน้าที่เป็น “เมนูหลัก” (Main Menu) สำหรับให้ผู้ใช้เลือกการทำงานต่าง ๆ ของระบบขายรถยนต์มือสอง **JB Garage** โดยในฟังก์ชันนี้จะมีการวนลูปเพื่อให้ผู้ใช้สามารถทำงานซ้ำได้หลายครั้งจนกว่าจะเลือก “ออกจากโปรแกรม (Exit)”

2.1 ส่วนที่ 1: การเรียกใช้โมดูลฟังก์ชันอื่น

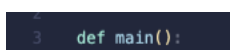


ภาพที่ 2-1 ภาพการ Import Function

บรรทัดนี้เป็นการนำเข้าไฟล์ **function.py** ซึ่งเก็บฟังก์ชันย่อยต่าง ๆ ของโปรแกรม เช่น `Add()`, `Update()`, `Delete()`, และ `View()`

การใช้คำสั่ง `as md` เป็นการตั้งชื่อย่อให้กับโมดูล เพื่อให้เรียกใช้งานได้สั้นลง เช่น `md.Add()` แทนการเขียน `function.Add()`

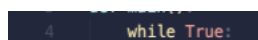
2.2 ส่วนที่ 2: การกำหนดฟังก์ชันหลัก



ภาพที่ 2-2 Function Main ()

กำหนดฟังก์ชันชื่อ `main()` เพื่อเป็นจุดเริ่มต้นของการทำงานทั้งหมดของโปรแกรม ภายในฟังก์ชันนี้จะควบคุมลูปการแสดงผลเมนูและการเลือกเมนูย่อยของผู้ใช้ทั้งหมด

2.3 ส่วนที่ 3: การวนลูปแสดงเมนูหลัก



ภาพที่ 2-3 การวนลูปแสดงเมนูหลัก

ใช้ลูป `while True` เพื่อให้เมนูหลักแสดงซ้ำไปเรื่อย ๆ โปรแกรมจะวนกลับมาที่เมนูนี้ทุกครั้ง หลังจากผู้ใช้ดำเนินการเสร็จในแต่ละคำสั่ง ลูปจะสิ้นสุดเมื่อผู้ใช้เลือก “Exit (ข้อ 5)”

ส่วนที่ 4: การรับค่าจากผู้ใช้และตรวจสอบข้อผิดพลาด

```

5      try:
6          print('---- Welcome to JB Garage Used Car System ----')
7          Choice = input('    1.Add Car
8          2.Update Car
9          3.Delete Car
10         4.View Car
11         5.Exit
12         Enter : ')
13         print('-----')
14
15         if Choice not in ['1','2','3','4','5']:
16             print(' Error: ValueError!!')
17             continue
18
19     except ValueError:
20         print(' Error ValueError!!')
21         continue

```

ภาพที่ 2-4 การรับ input จากผู้ใช้ และการตรวจสอบ ValueError

แสดงข้อความต้อนรับและเมนูหลักให้ผู้ใช้เลือกการทำงาน

รับค่าที่ผู้ใช้กรอกผ่านคำสั่ง input() และเก็บไว้ในตัวแปร Choice

ใช้ try: เพื่อดักจับข้อผิดพลาด เช่น กรณีผู้ใช้กรอกค่าที่ไม่ใช่ตัวเลข 1-5

** ซึ่งหากไม่ใช่ เลข 1-5

ตรวจสอบว่าค่าที่ผู้ใช้กรอกเข้ามาเป็นหนึ่งในตัวเลือกที่ถูกต้องหรือไม่

หากไม่ถูกต้อง โปรแกรมจะแสดงข้อความ “Error: ValueError!!” และกลับไปเริ่มเมนูใหม่ (continue)

2.4 ส่วนที่ 6: การใช้ match-case เพื่อเลือกเมนูที่ต้องการ

```

23      match Choice:
24          case '1':
25              md.Add()
26          case '2':
27              md.Update()
28          case '3':
29              md.Delete()

```

ภาพที่ 2-5 การใช้ Match – Case

- ใช้คำสั่ง match-case (ลักษณะเดียวกับ switch-case ในภาษาอื่น)
- เรียกใช้ฟังก์ชันที่เกี่ยวข้องกับตัวเลือกของผู้ใช้ เช่น
 - case '1' เรียก md.Add() → เพิ่มข้อมูลรถยนต์
 - case '2' เรียก md.Update() → แก้ไขข้อมูลรถยนต์
 - case '3' เรียก md.Delete() → ลบข้อมูลรถยนต์

2.5 ส่วนที่ 7: เมนูย่อยสำหรับ “ดูข้อมูลรถยนต์”

```

30         case '4':
31             while True:
32                 try:
33                     view_choice = input('
34                     1.Single Car
35                     2.All Cars
36                     3.Filter
37                     4.Car not sale + Summary
38                     5.Sold Car (with customer + Summary)
39                     6.Exit
40                     Enter : ')
41                     print('-----')
42                     if view_choice in ['1','2','3','4','5']:
43                         md.View(int(view_choice))
44                     elif view_choice == '6':
45                         break
46                     else:
47                         print(' Error: ValueError!!')
48                         print('-----')
49                     except ValueError:
50                         print(' Error: ValueError!!')
51         case '5':
52             print(' Thank you for using JB Garage Used Car System')
53             break

```

ภาพที่ 2-6 Choice ในการดูข้อมูลรถยนต์

เมื่อผู้ใช้เลือก “View Car” โปรแกรมจะเข้าสู่เมนูย่อยเพื่อเลือกวิธีการแสดงข้อมูลรถ

ผู้ใช้สามารถเลือกรถเพียงคันเดียว, รถทั้งหมด, รถที่ยังไม่ขาย หรือรถที่ขายแล้ว

ถ้าเลือก “6.Exit” จะออกจากเมนูย่อยและกลับสู่เมนูหลัก

ถ้าผู้ใช้เลือกตัวเลข 1-5 → จะเรียกใช้ฟังก์ชัน md.View() พร้อมส่งค่าหมายเลขตัวเลือกเข้าไป

ถ้าเลือก “6” → ใช้คำสั่ง break เพื่อออกจากเมนูย่อย

หากกรอกค่าผิด → จะแสดงข้อความ “Error: ValueError!!”

เมื่อผู้ใช้เลือกข้อ 5 “Exit” โปรแกรมจะแสดงข้อความขอบคุณและใช้คำสั่ง break เพื่อออกจากลูป

หลัก → สิ้นสุดการทำงานของโปรแกรม

2.6 ส่วนสุดท้าย จุดเริ่มต้นของโปรแกรม

```

55 if __name__ == '__main__':
56     main()
57

```

ภาพที่ 2-7 การเรียกใช้ Main()

บรรทัดนี้ทำหน้าที่ให้ Python เริ่มรันโปรแกรมจากฟังก์ชัน main()

ถ้าไฟล์นี้ถูกนำไป import โดยไฟล์อื่น โปรแกรมจะไม่รันทันทีจนกว่าจะถูกเรียกใช้งานโดยตรง

สรุปภาพรวมการทำงานของฟังก์ชัน main()

| ลำดับ | ขั้นตอน | รายละเอียด |
|-------|----------------------|--|
| 1 | แสดงเมนูหลัก | แสดงรายการคำสั่งให้ผู้ใช้เลือก |
| 2 | รับค่าจากผู้ใช้ | รับค่าเมนูจากคีย์บอร์ด |
| 3 | ตรวจสอบความถูกต้อง | ตรวจสอบว่าเป็นตัวเลข 1-5 |
| 4 | เรียกใช้ฟังก์ชันย่อย | เช่น เพิ่ม แก้ไข ลบ หรือดูข้อมูล |
| 5 | เมนูย่อย (View Car) | แสดงรูปแบบการดูข้อมูลรถยนต์หลายแบบ |
| 6 | ออกจากโปรแกรม | เมื่อผู้ใช้เลือก Exit โปรแกรมจะหยุดทำงาน |

ตารางที่ 2 สรุปการทำงานของฟังก์ชัน Main

บทที่ 3

การจัดการข้อมูลและการนำเข้าโมดูล (Import & Data Management)

ในบทนี้จะกล่าวถึงการจัดการข้อมูลของระบบขายรถยนต์มือสอง JB Garage ซึ่งมีการนำเข้าฟังก์ชันและโมดูลต่าง ๆ เพื่อช่วยให้ระบบสามารถจัดเก็บ ดึงข้อมูล และแปลงข้อมูลให้อยู่ในรูปแบบที่เหมาะสมสำหรับการประมวลผลได้อย่างมีประสิทธิภาพ โดยโครงสร้างข้อมูลหลักของระบบจะถูกจัดเก็บในไฟล์ **binary (.dat)** เพื่อให้การเข้าถึงข้อมูลมีความรวดเร็วและปลอดภัย

3.1 การนำเข้าโมดูล (Import Modules)

ในตอนต้นของไฟล์ function.py มีการนำเข้าโมดูลสำคัญ 3 ตัว ดังนี้



```
function.py > ...
1 import struct, os, re
```

ภาพที่ 3-1 Module ที่ import

3.1.1 Struct

ใช้สำหรับจัดการข้อมูลให้อยู่ในรูปแบบ **Binary Format** โดยใช้

คำสั่ง struct.pack() และ struct.unpack() เพื่อแปลงข้อมูลจากชนิดข้อมูลปกติ (เช่น string, int, float) ให้เป็น byte ก่อนจัดเก็บในไฟล์ และสามารถแปลงกลับมาได้เมื่อต้องการอ่านข้อมูล

3.1.2 Os

ใช้ตรวจสอบและจัดการไฟล์ เช่น ตรวจสอบว่าไฟล์มีอยู่จริงหรือไม่ (os.path.exists()), เปิด-เขียนไฟล์ในโหมด binary (wb, rb), และช่วยให้ระบบสามารถทำงานข้ามระบบปฏิบัติการได้อย่างถูกต้อง

3.1.3 re (Regular Expression)

ใช้ตรวจสอบความถูกต้องของข้อมูลที่ใช้ป้อนเข้ามา เช่น การตรวจสอบรหัสรถยนต์ให้มีรูปแบบ "C001" ด้วยคำสั่ง re.fullmatch(r"C\d{3}", car_id)

3.2 การกำหนดชื่อไฟล์เก็บข้อมูล

| ไฟล์ | รายละเอียด |
|------------------------------|---|
| <code>cars_basic.dat</code> | ใช้เก็บข้อมูลพื้นฐานของรถ เช่น รหัสรถ ยี่ห้อ รุ่น ปี และราคาซื้อ |
| <code>cars_status.dat</code> | ใช้เก็บสถานะของรถ เช่น ยังขายอยู่หรือขายแล้ว พร้อมราคาขายที่ตั้งไว้ |
| <code>cars_sale.dat</code> | ใช้เก็บข้อมูลการขาย เช่น ราคาสุดท้ายที่ขายจริง และข้อมูลลูกค้า |

ตารางที่ 3 ข้อมูลที่แต่ละ File จัดเก็บ

3.2.1 การเชื่อมโยงข้อมูลระหว่างไฟล์

`car_id` เหมือนเป็น รหัสตัวแทนรถคันนั้น

ถ้าเราจะรู้ข้อมูลว่า “รถคันนี้ขายให้ใคร” → ต้องไปอ่านจาก `cars_sale.dat` เพราะ ชื่อผู้ซื้อและเบอร์โทรจะอยู่ในไฟล์นี้เท่านั้น

`cars_status.dat` จะบอกว่า “ขายแล้วหรือยัง” (`is_sold`) และราคาที่ตั้งไว้ (`sell_price`)

`cars_basic.dat` จะบอกรายละเอียดพื้นฐาน เช่น ยี่ห้อ, รุ่น, ปี, ราคาซื้อ

```

3 # ===== File Names =====
4 FILE_BASIC = "cars_basic.dat"
5 FILE_STATUS = "cars_status.dat"
6 FILE_SALE = "cars_sale.dat"
7
8 # =====

```

ภาพที่ 3-2 ชื่อไฟล์ที่จัดเก็บข้อมูล ทั้ง 3 ไฟล์

3.3 การกำหนดโครงสร้างข้อมูลด้วย Struct

เมื่อเราจะเก็บข้อมูลรถแต่ละคันลงไฟล์ `.dat`:

- ข้อมูลแต่ละคันมีหลายฟิลด์ เช่น รหัสรถ, ยี่ห้อ, รุ่น, ปี, ราคาซื้อ, ราคาขาย ฯลฯ
- บางฟิลด์เป็น **ตัวเลข** (int/float) บางฟิลด์เป็น **ข้อความ** (string)
- **struct** ช่วยให้เราทำ 2 สิ่งสำคัญได้ง่าย:
 1. **Pack** → แปลง Python data → ไบต์ เพื่อเก็บในไฟล์ binary
 2. **Unpack** → แปลงไบต์จากไฟล์ → Python data

ข้อดี:

- ข้อมูลในไฟล์ขนาดเล็ก เพราะเป็น binary
- อ่านเขียนเร็ว
- ฟอแมตชัดเจน ใช้ร่วมกับหลายภาษาได้ (ถ้าใช้ byte order เดียวกัน)

โครงสร้างโปรเจกต์เรา

3.3.1 Struct_Basic

```

12 # 1. Basic: เก็บข้อมูลพื้นฐานของรถแต่ละคันใน cars_basic.dat
13 #   - car_id (int)       : รหัสรถ (ใช้เป็น key เชื่อมโยงทุกไฟล์)
14 #   - year (int)        : ปีที่ผลิต
15 #   - brand (20s)       : ยี่ห้อรถ (string 20 bytes)
16 #   - model (20s)       : รุ่นรถ (string 20 bytes)
17 #   - odometer (int)     : เลขไมล์
18 #   - buy_price (int)    : ราคาซื้อ
19 struct_basic = struct.Struct("<i i 20s 20s i i")

```

ภาพที่ 3-3 Struct_basic

| Index | ชื่อฟิลด์ | Format | ความหมาย |
|-------|-----------|--------|----------------------------|
| 0 | car_id | i | รหัสรถ (int) |
| 1 | year | i | ปีผลิต (int) |
| 2 | brand | 20s | ยี่ห้อรถ (string 20 bytes) |
| 3 | model | 20s | รุ่นรถ (string 20 bytes) |
| 4 | odometer | i | เลขไมล์ (int) |
| 5 | buy_price | i | ราคาซื้อ (int) |

ตารางที่ 4 ข้อมูลในตาราง Struct_Basic

3.3.2 Struct_Status

```

21 # 2. Status: เก็บสถานะของรถแต่ละคันใน cars_status.dat
22 #   - car_id (int)       : รหัสรถ (key เชื่อมโยงกับ cars_basic.dat)
23 #   - active (int)       : สถานะการใช้งาน (1=active, 0=inactive)
24 #   - is_sold (int)      : ขายแล้วหรือยัง (1=ขายแล้ว, 0=ยังไม่ขาย)
25 #   - sell_price (float) : ราคาขายที่ตั้งไว้
26 struct_status = struct.Struct("<i i i f")

```

ภาพที่ 3-4 Struct_Status

| ลำดับ | ชื่อฟิลด์ | Format | ความหมาย |
|-------|------------|--------|--------------------------------------|
| 0 | car_id | l | รหัสรถ |
| 1 | active | l | สถานะการใช้งาน 1=active,0=inactive |
| 2 | is_sold | l | ขายแล้วหรือยัง 1=ขายแล้ว,0=ยังไม่ขาย |
| 3 | sell_price | f | ราคาขายที่ตั้งไว้ (float) |

ตารางที่ 5 ข้อมูลที่จัดเก็บใน Struct_Status

- f = float 32-bit
- ใช้เก็บราคาที่ตั้งไว้สำหรับรถคันนั้น

3.4 ฟังก์ชันช่วยเหลือในการจัดการข้อความ

ในการจัดเก็บข้อมูลลงไฟล์แบบ binary ด้วย struct ข้อความ (string) ต้องมีขนาดตายตัว (fixed size) เพื่อให้การ pack/unpack ถูกต้อง ดังนั้นโปรเจกต์จึงมีฟังก์ชันช่วยเหลือสำหรับการจัดการ string ดังนี้

```
# ===== Helper =====
def encode_str(s, size):
    return s.encode("utf-8")[:size].ljust(size, b'\x00')

def decode_str(b):
    return b.split(b'\x00', 1)[0].decode("utf-8")
```

ภาพที่ 3-5 โค้ดที่ช่วยในการจัดการตาราง และการ Encode

หน้าที่:

แปลงข้อความ Python (str) เป็น bytes (utf-8)

ตัดความยาวให้ไม่เกินขนาดที่กำหนด

เติม null byte (b'\x00') ให้ครบขนาด field

3.5 การบันทึกข้อมูลทั้งหมด (Save All)

ฟังก์ชัน `save_all` เป็นฟังก์ชันหลักสำหรับโปรเจกต์ ระบบขายรถยนต์มือสอง มีหน้าที่ เขียนข้อมูลรถทั้งหมดลงไฟล์ `binary` ทั้ง 3 ไฟล์ (`cars_basic.dat`, `cars_status.dat`, `cars_sale.dat`) โดยใช้โครงสร้าง `struct` และฟังก์ชันช่วยเหลือ `encode_str` เพื่อจัดการ `string` ขนาดคงที่

3.5.1 วัตถุประสงค์

- จัดเก็บ ข้อมูลรถแต่ละคัน อย่างเป็นระบบ
- แยกข้อมูลตาม ประเภทและการใช้งาน
 - o `cars_basic.dat` → ข้อมูลพื้นฐานรถ
 - o `cars_status.dat` → สถานะการขาย/การใช้งาน
 - o `cars_sale.dat` → ข้อมูลการขายจริงและลูกค้า
- ทำให้ อ่าน/เขียนเร็ว เพราะเป็น `binary data`
- ลดข้อผิดพลาดจากการจัดการ `string` ขนาดไม่เท่ากัน

3.5.2 การเปิดไฟล์

```
48 def save_all(cars):
49     with open(FILE_BASIC, "wb") as fb, open(FILE_STATUS, "wb") as fs, open(FILE_SALE, "wb") as fsl:
```

ภาพที่ 3-6 การเปิดไฟล์

- เปิด 3 ไฟล์พร้อมกัน
- `"wb"` → `write binary`
- การเปิดไฟล์แบบนี้ทำให้ไฟล์เดิม ถูกเขียนทับทั้งหมด
- `fb`, `fs`, `fsl` เป็น `handle` สำหรับเขียนลงไฟล์แต่ละประเภท

เทคนิคเชิงลึก: การเปิดหลายไฟล์พร้อมกันใน `with` ทำให้ ไม่ต้องปิดไฟล์ทีละไฟล์ Python จะปิดไฟล์ให้เองเมื่อออกจาก `with`

การวนลูปข้อมูลรถ

```
50 for c in cars:
51     car_id_int = int(c["car_id"][1:]) # remove 'C' → int
```

ภาพที่ 3-7 Loop Cars Information

- วนลูปทุก dictionary ใน cars
- แปลง CarID เช่น "C001" → 1
- จำเป็นเพราะ struct ต้องใช้ **int** แทน string

เทคนิคเชิงลึก: การเก็บ CarID เป็น "Cxxx" ทำให้ผู้ใช้จำง่าย แต่ในการบันทึกไฟล์ ต้องแปลงเป็น **int** เพื่อประสิทธิภาพและลดขนาดไฟล์

3.5.3 การบันทึกไฟล์แต่ละประเภท

บันทึกข้อมูลพื้นฐาน (cars_basic.dat)

```
car_id_int = int(c["car_id"][1:]) * 1000
# BASIC
fb.write(struct_basic.pack(
    car_id_int,
    int(c["year"]),
    encode_str(c["brand"], 20),
    encode_str(c["model"], 20),
    int(c["odometer"]),
    int(c["buy_price"])
))
```

ภาพที่ 3-8 บันทึกไฟล์ข้อมูลพื้นฐาน *basic.dat*

- struct_basic: <i i 20s 20s i i
- i → int 4 byte
- 20s → string 20 byte
- String ต้องแปลงเป็น bytes ด้วย encode_str
- ข้อมูลที่ pack → เขียนลงไฟล์แบบ **binary**

บันทึกสถานะรถ (cars_status.dat)

```
# STATUS
fs.write(struct_status.pack(
    car_id_int,
    1, # active
    1 if c["status"].lower() == "yes" else 0,
    float(c["sell_price"])
))
```

ภาพที่ 3-9 บันทึกสถานะรถ

- struct_status: <i i i f

- $i \rightarrow \text{int 4 byte}$
- $f \rightarrow \text{float 4 byte}$
- สถานะ active ตั้งเป็น 1 เสมอ
- สถานะขาย (is_sold) แปลง Yes/No $\rightarrow 1/0$
- ราคาขาย (sell_price) แปลงเป็น float

บันทึกข้อมูลขายจริง (cars_sale.dat)

```
# SALE
fsl.write(struct_sale.pack(
    car_id_int,
    float(c["buy_price"]),
    float(c["sell_price"]),
    float(c["final_price"]),
    encode_str(c["customer_name"], 30),
    encode_str(c["customer_phone"], 15)
))
```

ภาพที่ 3-10 บันทึกข้อมูลการขายจริง

- **struct_sale:** <i f f f 30s 15s
 - เก็บราคาซื้อ, ราคาขาย, ราคาสุดท้าย
 - ชื่อลูกค้า (customer_name) $\rightarrow 30 \text{ byte}$
 - เบอร์โทร (customer_phone) $\rightarrow 15 \text{ byte}$
- String ต้องใช้ encode_str ทุกครั้ง

3.5.4 เทคนิคและข้อควรระวัง

1. **Binary Data Fixed Size:**
 - o ทุก field ต้องมีขนาดตายตัว
 - o String ต้องเติม null byte (b'\x00')
 - o Integer/float ต้องตรงชนิดตาม struct
2. ลำดับการ pack ต้องตรงกับ struct:
 - o ถ้าสลับฟิลด์ \rightarrow unpack ผิด \rightarrow ข้อมูลเสียหาย
3. เขียนทับไฟล์เดิม:

- "wb" → ข้อมูลเก่าหายหมด
- ต้องใช้ร่วมกับ load_all หากแก้ไขข้อมูล

4. Error Handling (เพิ่มเติม):

- ในระบบจริงควรใช้ try/except เพื่อจับข้อผิดพลาดเช่นไฟล์ไม่สามารถเปิดหรือ write ผิดพลาด

ภาพรวมและ Flow การทำงาน

Python Dict (cars)

|



for c in cars:

└─> encode_str (Brand, Model, Customer Name, Phone)

└─> struct_basic.pack → write to cars_basic.dat

└─> struct_status.pack → write to cars_status.dat

└─> struct_sale.pack → write to cars_sale.dat

- ทุกคันรถถูกเขียนลง 3 ไฟล์พร้อมกัน
- ข้อมูลพร้อมอ่านกลับด้วย load_all

| CarID | Brand | Model | Year | Odometer | BuyPrice | SellPrice | Status | FinalPrice | CustomerName | CustomerPhone |
|-------|--------|-------|------|----------|----------|-----------|--------|------------|--------------|---------------|
| C001 | Toyota | Vios | 2022 | 15000 | 550000 | 600000 | Yes | 600000 | John Doe | 0812345678 |

ตารางที่ 6 ข้อมูลจริงใน File

- หลัง save_all(cars) ข้อมูลจะถูกแยกเก็บเป็น **binary fixed size**
- เมื่อใช้ load_all() → ได้ dict เดิมครบทุกฟิลด์

| Report: Car Sold with Customer | | | | | | | | | | | |
|--------------------------------|-------|-------|------|--------------|-----------|------------|------|-------------|--------|---------------|----------------|
| CarID | Brand | Model | Year | Odometer(km) | Buy Price | Sell Price | Sold | Final Price | Profit | Customer Name | Customer Phone |
| C002 | To | Yo | 2019 | 200 | 50.00 | 500.00 | Yes | 500.00 | 450.00 | JayGumDae | 191 |

ภาพที่ 3-11 ตารางเก็บข้อมูลที่ Report อิงจาก Not Sale.txt

3.6 การโหลดข้อมูลทั้งหมด (Load All)

ฟังก์ชัน `load_all` เป็นฟังก์ชันที่ใช้ อ่านข้อมูลทั้งหมดจากไฟล์ binary ทั้ง 3

ไฟล์ (`cars_basic.dat`, `cars_status.dat`, `cars_sale.dat`) และรวมเป็น list ของ

dictionary เพื่อให้โปรแกรมนำไปใช้งานต่อได้ เช่น แสดงข้อมูล, คำนวณกำไร, หรือสร้างรายงาน

3.6.1 วัตถุประสงค์

- ☐ อ่านข้อมูลที่บันทึกไว้ในไฟล์ binary
- ☐ แปลง `bytes` → Python data type (int, float, string)
- ☐ รวมข้อมูลจาก 3 ไฟล์เป็น dict เดี่ยวต่อรถแต่ละคัน
- ☐ คำนวณค่า profit อัตโนมัติ

3.6.2 การตรวจสอบไฟล์

```
def load_all():
    cars = []
    if not (os.path.exists(FILE_BASIC) and os.path.exists(FILE_STATUS) and os.path.exists(FILE_SALE)):
        return []
```

ภาพที่ 3-12 การตรวจสอบ File

- เช็คว่า ไฟล์ทุกไฟล์มีอยู่จริง
- ถ้าไฟล์ใดหาย → คืนค่า list ว่าง
- ป้องกัน `FileNotFoundError`

เทคนิคเชิงลึก: การเช็คไฟล์ก่อนอ่านช่วยให้โปรแกรม **robust** และไม่ crash

3.7 การเปิดไฟล์

```
85 with open(FILE_BASIC, "rb") as fb, open(FILE_STATUS, "rb") as fs, open(FILE_SALE, "rb") as fsl:
```

ภาพที่ 3-13 การเปิดไฟล์

- ☐ เปิดไฟล์ 3 ไฟล์พร้อมกัน แบบ read binary (rb)

- ☐ ตัวแปร fb, fs, fsl เป็น handle สำหรับอ่านไฟล์แต่ละประเภท
- ☐ ใช้ with เพื่อให้ไฟล์ ถูกปิดอัตโนมัติ หลังใช้งาน

3.7.1 การวนลูปอ่านข้อมูลที่ละบล็อก

```
while True:
    cb = fb.read(struct_basic.size)
    cs = fs.read(struct_status.size)
    cl = fsl.read(struct_sale.size)
    if not cb or not cs or not cl:
        break
```

ภาพที่ 3-14 การวนค่าทีละ Block

- อ่านไฟล์ทีละบล็อก ขนาดเท่ากับ struct ของแต่ละไฟล์
 - struct_basic.size, struct_status.size, struct_sale.size
- ถ้าเจอ EOF (ไม่อ่านได้แล้ว) → break

เทคนิคเชิงลึก: การอ่านทีละ block ทำให้ **memory efficient** แม้มีรูดจำนวนมาก

3.7.2 การ unpack ข้อมูล

```
break
b = struct_basic.unpack(cb)
s = struct_status.unpack(cs)
l = struct_sale.unpack(cl)
```

ภาพที่ 3-15 การ Unpack ข้อมูล

- แปลง bytes → tuple ของค่าตาม struct
- b → ข้อมูล basic
- s → ข้อมูล status
- l → ข้อมูล sale

3.7.3 การรวมข้อมูลเป็น dictionary

```

95         car_id = f"C{b[0]:03d}"
96         cars.append({
97             "car_id": car_id,
98             "year": b[1],
99             "brand": decode_str(b[2]),
100            "model": decode_str(b[3]),
101            "odometer": b[4],
102            "buy_price": b[5],
103            "status": "Yes" if s[2] == 1 else "No",
104            "sell_price": s[3],
105            "final_price": l[3],
106            "profit": l[3] - b[5] if l[3] > 0 else 0,
107            "customer_name": decode_str(l[4]),
108            "customer_phone": decode_str(l[5])
109        })
110     return cars

```

ภาพที่ 3-16 การรวมข้อมูลทั้งหมด เพื่อนำไปใช้ต่อ

- CarID:
 - o แปลงจาก int → string format "Cxxx"
 - o เช่น 1 → "C001"
- String fields:
 - o ใช้ decode_str แปลง bytes → Python string
 - o เช่น Brand, Model, Customer Name, Customer Phone
- Status:
 - o ถ้า is_sold = 1 → "Yes"
 - o ถ้า 0 → "No"
- Profit:
 - o คำนวณ final_price - buy_price
 - o ถ้ายังไม่ขาย → profit = 0
- คำนวณค่าเป็น list ของ dict
- แต่ละ dict แทนรถ 1 คัน พร้อมข้อมูล ครบทุก field
- ใช้ต่อได้ทั้ง View, Report, Edit, Delete

Flow Diagram (ภาพรวม)

[cars_basic.dat] → struct_basic.unpack → Basic info

[cars_status.dat] → struct_status.unpack → Status info

[cars_sale.dat] → struct_sale.unpack → Sale info



3.7.4 ตัวอย่างผลลัพธ์ของ load_all()

```
[{  
    'car_id': 'C001',      'year': 2022,    'brand': 'Toyota',  
    'model': 'Vios',      'odometer': 15000,    'buy_price': 550000,  
    'status': 'Yes',      'sell_price': 600000,    'final_price': 600000,  
    'profit': 50000,      'customer_name': 'John Doe', 'customer_phone': '0812345678'  
},...]
```

สามารถนำ dict นี้ไป แสดง, ลบ, แก้ไข หรือทำรายงาน ได้โดยตรง

3.8 สรุปภาพรวมของการจัดการข้อมูล

3.8.1 ภาพรวมของการจัดการข้อมูล

ในระบบ JB Garage ข้อมูลรถถูกเก็บใน 3 ไฟล์ binary ได้แก่

| ไฟล์ | ประเภทข้อมูล |
|-----------------|---|
| cars_basic.dat | ข้อมูลพื้นฐาน: car_id, year, brand, model, odometer, buy_price |
| cars_status.dat | สถานะรถ: car_id, active, is_sold, sell_price |
| cars_sale.dat | ข้อมูลการขาย: car_id, buy_price, sell_price, final_price, customer_name, customer_phone |

ตารางที่ 7 ข้อมูลที่เก็บเป็น Binary

3.8.2 หลักการจัดการข้อมูล:

1. Struct → กำหนดโครงสร้าง binary fixed-size
2. ฟังก์ชันช่วยเหลื string → encode_str / decode_str
3. Save All → เขียนข้อมูล dict ของรถลงไฟล์ binary ทั้ง 3 ไฟล์
4. Load All → อ่านไฟล์ binary → unpack → รวมเป็น dict ต่อคัน
5. Add/Delete → เพิ่มหรือลบรถ → เรียก save_all
6. View / Reports → โหลดข้อมูล → แสดงรายละเอียด / ตาราง / รายงานไฟล์ .txt

ข้อดี:

- แม่นยำ, memory-efficient, อ่าน/เขียนเร็ว
- พร้อมใช้งานทุกฟังก์ชัน เช่น เพิ่ม ลบ แก้ไข ดู และสร้างรายงาน

ภาพรวม flow การทำงาน:

User Input → Dictionary → Save All → Binary Files

Binary Files → Load All → List of Dict → View / Report / Add / Delete

ระบบนี้ทำให้ข้อมูลครบถ้วน ถูกต้อง และพร้อมใช้งานทุกฟังก์ชันของโปรแกรม

บทที่ 4

การอธิบายการทำงานของ Function Add

ฟังก์ชัน Add() ทำหน้าที่ รับข้อมูลรถใหม่จากผู้ใช้, ตรวจสอบความถูกต้อง และบันทึกลงไฟล์ โดยมีขั้นตอนหลักดังนี้:

4.1.1 โหลดข้อมูลเดิม

```
try:
    cars = load_all()
```

ภาพที่ 4-1 โหลดรถทั้งหมดเป็น list ของ dict เพื่อใช้ตรวจสอบ CarID ซ้ำ

4.1.2 กรอกและตรวจสอบข้อมูลรถ

- ☐ CarID: ต้องไม่ซ้ำและอยู่ในรูปแบบ Cxxx
- ☐ Brand / Model: ต้องไม่เว้นว่าง
- ☐ Year: เป็นตัวเลข 1900-2100
- ☐ Odometer: เป็นตัวเลขเท่านั้น
- ☐ Buy Price / Sell Price / Final Price: ต้องเป็นตัวเลขบวก
- ☐ Status: Yes หรือ No
- ☐ Customer Name / Phone: กรอกเฉพาะรถที่ขายแล้ว

4.1.3 สร้าง dictionary รถใหม่

```
# --- Create Car Dict ---
car = {
    "car_id": car_id,
    "brand": brand,
    "model": model,
    "year": year,
    "odometer": odometer,
    "buy_price": buy_price,
    "sell_price": sell_price,
    "status": status_inp.capitalize(),
    "final_price": final_price,
    "profit": final_price - buy_price if final_price > 0 else 0,
    "customer_name": cname,
    "customer_phone": cphone
}
```

ภาพที่ 4-2 สร้าง Dictionary รถใหม่

4.1.4 เพิ่มลง list ของรถและบันทึก

```
cars.append(car)
save_all(cars)
print("Car added successfully!")
```

ภาพที่ 4-3 เพิ่มข้อมูลลงใน List ของรถและบันทึก

ใช้ save_all() เขียนข้อมูลลง 3 ไฟล์ binary (Basic / Status / Sale) พร้อมกัน

4.1.5 จัดการข้อผิดพลาด

ใช้ try-except เพื่อจับข้อผิดพลาดที่ไม่คาดคิด

4.1.6 ภาพรวมการทำงาน และกระบวนการของ Function Update

- ส่วนเริ่มต้นและค้นหา CarID

```
1 def Update():
2     try:
3         cars = load_all()
4         car_id = input("Enter CarID to update: ").strip().upper()
5         found = False
6
7         for car in cars:
8             if car["car_id"] == car_id:
9                 found = True
10                print(f"Updating {car_id}...")
```

ภาพที่ 4-4 ฟังก์ชัน Update()

โหลดข้อมูลทั้งหมด รับ CarID จากผู้ใช้ ค้นหาว่ามีอยู่ในระบบหรือไม่ ถ้าเจอจะเริ่มอัปเดต

- กรณีรถยังไม่ได้ขาย

```

1  if car["status"].lower() == "no":
2      print("Currently not sold → mark SOLD")
3      car["status"] = "Yes"
4
5      # Final Price
6      while True:
7          try:
8              final_price = float(input("Final Price: "))
9              if final_price < 0:
10                 print("Error: Final Price cannot be negative!")
11                 continue
12                 car["final_price"] = final_price
13                 break
14             except ValueError:
15                 print("Error: Final Price must be a number!")
16
17         # Customer Name
18         while True:
19             cname = input("Customer Name: ").strip()
20             if cname == "":
21                 print("Error: Customer Name cannot be empty!")
22                 continue
23                 car["customer_name"] = cname
24                 break
25
26         # Customer Phone
27         while True:
28             cphone = input("Customer Phone: ").strip()
29             if not cphone.isdigit():
30                 print("Error: Phone must contain digits only!")
31                 continue
32                 if len(cphone) < 8 or len(cphone) > 15:
33                     print("Error: Phone length must be 8-15 digits!")
34                     continue
35                     car["customer_phone"] = cphone
36                     break

```

ภาพที่ 4-5 กรณีรถยังไม่ได้ขาย

ถ้ารถยัง **ไม่ขาย** จะเปลี่ยนสถานะเป็นขายแล้ว (Yes) และบังคับให้กรอกราคาขายจริง + ข้อมูลลูกค้า

- กรณีรถขายแล้ว อัปเดตข้อมูล

```

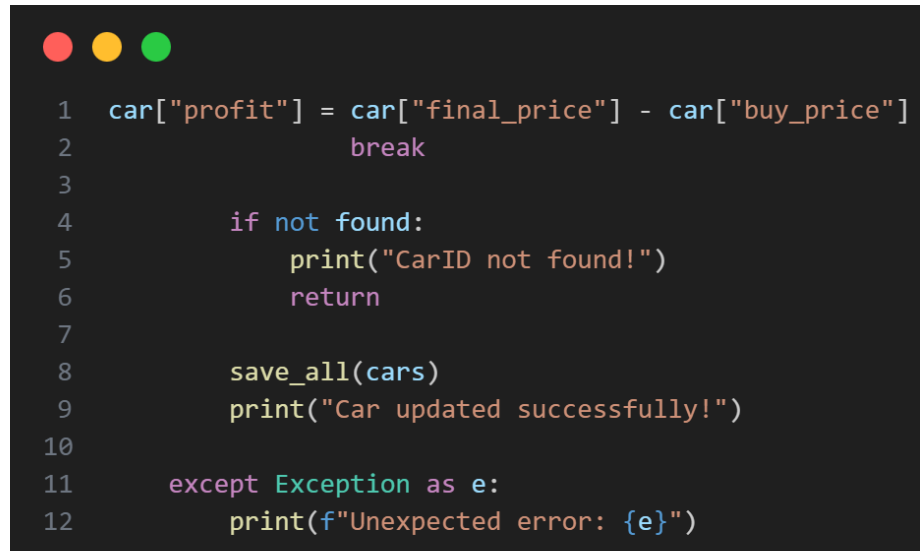
1  if car["status"].lower() == "no":
2      print("Currently not sold → mark SOLD")
3      car["status"] = "Yes"
4
5      # Final Price
6      while True:
7          try:
8              final_price = float(input("Final Price: "))
9              if final_price < 0:
10                 print("Error: Final Price cannot be negative!")
11                 continue
12                 car["final_price"] = final_price
13                 break
14             except ValueError:
15                 print("Error: Final Price must be a number!")
16
17         # Customer Name
18         while True:
19             cname = input("Customer Name: ").strip()
20             if cname == "":
21                 print("Error: Customer Name cannot be empty!")
22                 continue
23                 car["customer_name"] = cname
24                 break
25
26         # Customer Phone
27         while True:
28             cphone = input("Customer Phone: ").strip()
29             if not cphone.isdigit():
30                 print("Error: Phone must contain digits only!")
31                 continue
32                 if len(cphone) < 8 or len(cphone) > 15:
33                     print("Error: Phone length must be 8-15 digits!")
34                     continue
35                     car["customer_phone"] = cphone
36                     break

```

ภาพที่ 4-6 กรณีรถขายแล้ว อัปเดตข้อมูล

ถ้ารถถูกขายแล้ว สามารถแก้ไข Final Price, Customer Name, และ Customer Phone ได้ โดยใส่ข้อมูลใหม่หรือกด Enter เพื่อข้าม

- ส่วนสรุปและบันทึกไฟล์



```
1 car["profit"] = car["final_price"] - car["buy_price"]
2     break
3
4     if not found:
5         print("CarID not found!")
6         return
7
8     save_all(cars)
9     print("Car updated successfully!")
10
11 except Exception as e:
12     print(f"Unexpected error: {e}")
```

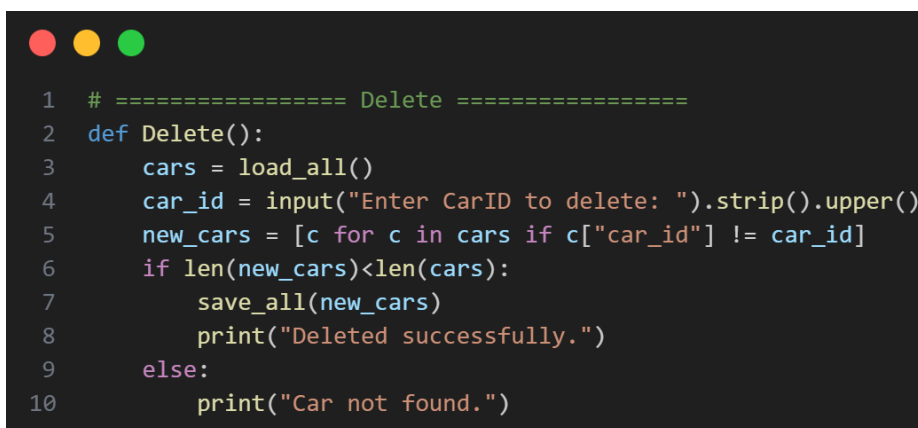
ภาพที่ 4-7 ส่วนสรุปและบันทึกไฟล์

หลังแก้ไขข้อมูลเสร็จ โปรแกรมจะคำนวณกำไรใหม่ บันทึกไฟล์ด้วย save_all() แสดงข้อความว่า การอัปเดตสำเร็จ

บทที่ 5

Function Delete

ฟังก์ชัน Delete() ใช้สำหรับ ลบข้อมูลรถยนต์ออกจากระบบ โดยผู้ใช้งานระบุ CarID ที่ต้องการลบ ระบบจะตรวจสอบว่ามีอยู่จริงหรือไม่ หากพบก็จะลบออกแล้วบันทึกข้อมูลใหม่ลงไฟล์



```

1  # ===== Delete =====
2  def Delete():
3      cars = load_all()
4      car_id = input("Enter CarID to delete: ").strip().upper()
5      new_cars = [c for c in cars if c["car_id"] != car_id]
6      if len(new_cars) < len(cars):
7          save_all(new_cars)
8          print("Deleted successfully.")
9      else:
10         print("Car not found.")

```

ภาพที่ 5-1 ฟังก์ชัน Delete()

โหลดข้อมูลรถทั้งหมดจากไฟล์ รับ CarID ที่ต้องการลบ สร้าง list ใหม่ที่ไม่มี CarID นั้น

ถ้า list ใหม่สั้นกว่าเดิม แสดงว่าลบสำเร็จ บันทึกไฟล์ใหม่ด้วย save_all() และแสดงข้อความ

ถ้า list ใหม่เท่าเดิม แสดงว่าไม่มี CarID ที่ระบุ แจ้งว่าไม่พบข้อมูล

ฟังก์ชันนี้ทำหน้าที่ลบข้อมูลรถออกจากระบบ โดยจะทำการสร้าง list ใหม่ที่ไม่มี CarID ที่ต้องการลบ

แล้วเปรียบเทียบกับข้อมูลเดิม หากมีการลบจริง ระบบจะบันทึกไฟล์ใหม่และแจ้งผลลัพธ์ ถ้าไม่มี

CarID นั้นอยู่ ระบบจะแจ้งว่าไม่พบข้อมูล

บทที่ 6

Function View ()

ฟังก์ชัน View() ใช้สำหรับ แสดงข้อมูลรถยนต์ ตามตัวเลือกที่ผู้ใช้เลือก โดยมีทั้งหมด 5 โหมด ได้แก่

1. แสดงรถเพียงคันเดียว (Single Car)
2. แสดงรถทั้งหมด (All Cars)
3. กรองข้อมูล (Filter)
4. แสดงรถที่ยังไม่ขาย + สรุป (Not Sold + Summary)
5. แสดงรถที่ขายแล้ว + สรุป (Sold + Summary)

- View(1) แสดงรถเพียงคันเดียว

```

1  def View(n:int):
2      cars = load_all()
3      if not cars:
4          print("No cars data.")
5          return
6
7      if n==1:
8          cid=input("Enter CarID: ").strip().upper()
9          for c in cars:
10             if c["car_id"]==cid:
11                 # แสดงข้อมูลรถเดียวแบบตาราง
12                 print("="*80)
13                 print(f"CarID : {c['car_id']}")
14                 print(f"Brand : {c['brand']}")
15                 print(f"Model : {c['model']}")
16                 print(f"Year : {c['year']}")
17                 print(f"Odometer : {c['odometer']:,} km")
18                 print(f"Buy Price : {c['buy_price']:,}")
19                 print(f"Sell Price: {c['sell_price']:,}")
20                 print(f"Status : {c['status']}")
21                 print(f"Final Price: {c['final_price']:,}")
22                 print(f"Profit : {c['profit']:,}")
23                 print(f"Customer : {c['customer_name']} ({c['customer_phone']})")
24                 print("="*80)
25                 return
26             print("Not found")

```

ภาพที่ 6-1 View(1) – แสดงรถเพียงคันเดียว

- View(2) แสดงรถทั้งหมด

```

1 elif n==2:
2     print("="*20, "All Cars", "="*20)
3     print(f"{'CarID':<6} | {'Brand':<10} | {'Model':<10} | {'Year':<6} | {'Status'}")
4     print("-"*50)
5     for c in cars:
6         print(f"{c['car_id']:<6} | {c['brand']:<10} | {c['model']:<10} | {c['year']:<6} | {c['status']}")
7     print("-"*50)

```

ภาพที่ 6-2 View(2) แสดงรถทั้งหมด

เมนูนี้จะแสดงรถยนต์ทั้งหมดในระบบในรูปแบบตารางสรุปสั้น ๆ ประกอบด้วย CarID, ยี่ห้อ, รุ่น, ปี และสถานะ

- View(3) กรองข้อมูล (Filter)

```

1 elif n==3:
2     print("=====")
3     print("Filter Options:")
4     print(" 1. Brand")
5     print(" 2. Model")
6     print(" 3. Year")
7     print(" 4. Status (Yes/No)")
8     choice = input("Enter choice: ").strip()
9
10    filtered = []
11    title = ""

```

ภาพที่ 6-3 View(3) กรองข้อมูล (Filter)

แสดงเมนูตัวเลือกการกรอง ให้ผู้ใช้เลือกว่าจะกรองตาม Brand, Model, Year หรือ Status

ถ้าเลือก 1. Brand โปรแกรมจะแสดงยี่ห้อที่มีในระบบ แล้วให้ผู้ใช้กรอกยี่ห้อที่ต้องการกรอง

```

1  if choice == "1":
2      brands = sorted(set(c['brand'] for c in cars))
3      print("\nAvailable Brand options:", ", ".join(brands))
4      print("-"*50)
5      brand = input("Enter Brand: ").strip().lower()
6      filtered = [c for c in cars if c['brand'].lower() == brand]
7      title = f"Cars Filtered by Brand = {brand.capitalize()}"
8

```

ภาพที่ 6-4 Filter brands

ถ้าเลือก 2. Model โปรแกรมจะแสดงรุ่นที่มีในระบบ แล้วให้ผู้ใช้กรอกรุ่นที่ต้องการกรอง

```

1  elif choice == "2":
2      models = sorted(set(c['model'] for c in cars))
3      print("\nAvailable Model options:", ", ".join(models))
4      print("-"*50)
5      model = input("Enter Model: ").strip().lower()
6      filtered = [c for c in cars if c['model'].lower() == model]
7      title = f"Cars Filtered by Model = {model.capitalize()}"
8

```

ภาพที่ 6-5 Filter Models

ถ้าเลือก 3. Year โปรแกรมจะแสดงปีทั้งหมด แล้วให้ผู้ใช้กรอกปีที่ต้องการกรอง

```

1  elif choice == "3":
2      years = sorted(set(str(c['year']) for c in cars))
3      print("\nAvailable Year options:", ", ".join(years))
4      print("-"*50)
5      year = input("Enter Year (YYYY): ").strip()
6      filtered = [c for c in cars if str(c['year']) == year]
7      title = f"Cars Filtered by Year = {year}"
8

```

ภาพที่ 6-6 Filter yeasers

ถ้าเลือก 4. Status โปรแกรมให้ผู้ใช้เลือกว่าจะกรองเฉพาะรถที่ขายแล้ว (Yes) หรือยังไม่ขาย (No)

ถ้าผู้ใช้ใส่ค่าที่ไม่ถูกต้อง ระบบจะแจ้งว่า “Invalid option!” และจบการทำงานของเมนูนี้

```

1 elif choice == "4":
2     print("\nAvailable Status options: Yes, No")
3     print("-"*50)
4     status = input("Enter Status (Yes/No): ").strip().lower()
5     filtered = [c for c in cars if c['status'].lower() == status]
6     title = f"Cars Filtered by Status = {status.capitalize()}"
7
8     else:
9         print("Invalid option!")
10        return
11

```

ภาพที่ 6-7 Filter status

แสดงผลลัพธ์ที่กรองได้ในรูปแบบตารางเหมือนเมนู View(2) ถ้าไม่พบข้อมูลก็จะขึ้นว่า “No cars found with this filter.”

```

1 # แสดงผลในรูปแบบตารางเหมือน All Cars
2 if filtered:
3     print(f"\n===== {title} =====")
4     print(f"{'CarID':<6} | {'Brand':<10} | {'Model':<10} | {'Year':<6} | {'Status'}")
5     print("-"*50)
6     for c in filtered:
7         print(f"{c['car_id']:<6} | {c['brand']:<10} | {c['model']:<10} | {c['year']:<6} | {c['status']}")
8     print("-"*50)
9 else:
10    print("No cars found with this filter.")

```

ภาพที่ 6-8 แสดงผลตาราง เหมือน All Cars

- View(4) รถที่ยังไม่ขาย + สรุป

```

1 elif n==4:
2     notsold=[c for c in cars if c['status'].lower()=="no"]
3     if not notsold:
4         print("No unsold cars.")
5         return
6     # แสดงบนจอแบบตาราง
7     print(make_table_not_sold(notsold,"Report: Car Not Sale"))
8     print(make_summary(cars,"Overall Summary"))
9     # เขียนไฟล์
10    report = make_table_not_sold(notsold,"Report: Car Not Sale")
11    report += "\n" + make_summary(cars,"Overall Summary")
12    with open("report_not_sale.txt","w",encoding="utf-8") as f:
13        f.write(report)
14    print("report_not_sale.txt generated.")
15

```

ภาพที่ 6-9 เมนู View(4)

เมนู View(4) จะกรองรถที่ยังไม่ได้ขาย (status = No) มาสร้างเป็นตารางสรุปโดยใช้ make_table_not_sold() จากนั้นจะเรียก make_summary() เพื่อสร้างสรุปจำนวนรถและสถิติอื่น ๆ และยังบันทึกออกเป็นไฟล์รายงาน report_not_sale.txt อีกด้วย

- View(5) รถที่ขายแล้ว + สรุป

เมนู View(5) จะกรองเฉพาะรถที่ขายแล้ว (status = Yes) มาสร้างเป็นตารางโดยใช้

```

1
2 elif n==5:
3     sold=[c for c in cars if c['status'].lower()=="yes"]
4     if not sold:
5         print("No sold cars.")
6         return
7     # แสดงบนจอแบบตาราง
8     print(make_table_sold(sold,"Report: Car Sold with Customer"))
9     print(make_summary(cars,"Sold Car Summary"))
10    # เขียนไฟล์
11    report = make_table_sold(sold,"Report: Car Sold with Customer")
12    report += "\n" + make_summary(cars,"Sold Car Summary")
13    with open("report_sold.txt","w",encoding="utf-8") as f:
14        f.write(report)
15    print("report_sold.txt generated.")

```

ภาพที่ 6-10 เมนู View(5)

บทที่ 7

การตกแต่ง

7.1 จุดเริ่มต้น: ปัญหาที่เราต้องแก้

เรามีข้อมูลรถสามไฟล์:

1. ข้อมูลพื้นฐาน (cars_basic.dat) → มีรหัสรถ, ยี่ห้อ, รุ่น, ปี, ไมล์, ราคาซื้อ
2. ข้อมูลสถานะ (cars_status.dat) → ว่ารถขายแล้วหรือยัง, ราคาที่ตั้งขาย
3. ข้อมูลการขายจริง (cars_sale.dat) → ราคาสุดท้าย, ชื่อและเบอร์ลูกค้า

ความยากตรงนี้ คือข้อมูลกระจายอยู่หลายไฟล์ เราต้องเอามารวมกันให้เป็นชุดเดียวกัน จะได้ทำงานได้

7.2 แปลงไฟล์เป็นข้อมูลที่โปรแกรมเข้าใจ

เราต้องอ่านไฟล์ทุกไฟล์และแปลงข้อมูล binary เป็น **list** ของ **dictionary** แต่ละ dict คือรถ 1 คัน

- เหมือนเรากำลังสร้าง "ฐานข้อมูลเล็ก ๆ ใน memory"
- ตอนนี้แต่ละ dict จะมีทุกข้อมูลที่เราต้องการ เช่น CarID, Brand, Model, Status, Final Price, Profit, Customer Name

ข้อดีคือ เราสามารถ filter, sort, หรือประมวลผลได้ง่าย

7.3 แยกรถตามสถานะ

- รถที่ยังไม่ขาย → สำหรับรายงาน Car Not Sale
- รถที่ขายแล้ว → สำหรับรายงาน Car Sold

เราต้องแยกก่อนเพราะ **column** ของแต่ละรายงานไม่เหมือนกัน

- รถยังไม่ขาย → ไม่ต้องมี Final Price, Profit, Customer Info
- รถขายแล้ว → ต้องมีทุกอย่าง

7.4 ออกแบบตารางให้อ่านง่าย

ความท้าทายอีกอย่าง คือเราต้องทำให้ตารางอ่านง่ายบน console หรือ export เป็นไฟล์ txt

- ต้องกำหนด **ความกว้างของ column** ล่วงหน้า
- ต้องมี **เส้นกรอบ** (+----+) เพื่อให้เป็นตารางสวย
- ข้อมูลตัวเลข เช่น ไมล์, ราคา → ต้องจัด **align** ขวา

- ข้อมูลตัวอักษร → align ซ้าย

ทำให้คนอ่านมองที่เดียวเข้าใจว่า column ไหนคืออะไร

7.5 สรุปข้อมูล (Summary)

นอกจากตารางแล้ว เราต้องมี สรุปภาพรวม:

1. จำนวนรถทั้งหมด
2. จำนวนรถขายแล้ว / ยังไม่ขาย
3. สถิติราคาขายจริง (Min, Max, Avg) สำหรับรถขายแล้ว
4. จำนวนรถแต่ละแบรนด์

เหตุผล: จะช่วยให้ผู้จัดการดูภาพรวมได้ทันทีโดยไม่ต้องไปนับเอง

7.6 ความซับซ้อนของงาน

- ต้องรวมข้อมูลหลายไฟล์
- ต้องแยกสถานะรถ
- ต้องสร้าง table แบบ align สวยทั้งตัวเลขและตัวอักษร
- ต้องทำ summary พร้อมสถิติและสรุปตามแบรนด์

นี่คือเหตุผลว่าทำไมโค้ดมันยาวและต้องมีหลายฟังก์ชัน แต่แต่ละส่วนมีเหตุผลชัดเจน

จนได้ ภาพหน้าตา Summary แบบนี้

```

1
2 Report: Car Not Sale
3
4 | CarID | Brand | Model | Year | Odometer(km) | Buy Price | Sell Price | Sold |
5 |-----|-----|-----|-----|-----|-----|-----|-----|-----|
6 | C001 | Toyo | So | 2019 | 50 | 20.00 | 10.00 | No |
7 |-----|-----|-----|-----|-----|-----|-----|-----|-----|
8
9 Overall Summary
10 * Total Cars : 1
11 * Sold Cars : 0
12 * Available : 1
13
14 Cars by Brand
15 * Toyo : 1

```

ภาพที่ 7-1 หน้าตา Summary ที่คาดหวัง

บทที่ 8

Summary และ การไหลของข้อมูล

8.1 การไหลของข้อมูลในรายงาน

8.1.1 Step 1: โหลดข้อมูล (load_all())

- **จุดประสงค์:** เอาข้อมูลจากไฟล์ทั้งสามมารวมเป็น list ของ dict
- **กระบวนการ:**
 1. เปิดไฟล์ cars_basic.dat, cars_status.dat, cars_sale.dat
 2. อ่านข้อมูลที่ละ record (ใช้ struct unpack)
 3. แปลง byte string เป็น string ปกติ (decode_str)
 4. สร้าง dict สำหรับแต่ละรถ → เก็บทุก attribute ที่จำเป็น
- **ผลลัพธ์:** list ของ dict ที่มีข้อมูลครบทุกคัน

8.1.2 Step 2: แยกข้อมูลตามสถานะ

- **เหตุผล:** ตารางของรถขายแล้ว กับ รถยังไม่ขายไม่เหมือนกัน
- **วิธีทำ:**
 - สร้างสอง list แยกจาก status field
 - "No" → รถยังไม่ขาย
 - "Yes" → รถขายแล้ว

8.1.3 Step 3: สร้างตาราง (make_table_not_sold() / make_table_sold())

- **จุดสำคัญ:** ต้องทำให้ตารางอ่านง่ายและสวยงาม console
- **วิธีคิด:**
 1. กำหนด ความกว้างของแต่ละ column ให้พอดีกับข้อมูล
 2. สร้าง เส้นกรอบบน-ล่าง (+-----+)
 3. สร้าง header → ชื่อ column
 4. วนลูปแต่ละรถ → สร้าง row
 5. รวม header + rows + เส้นกรอบปิดท้าย

- ความต่างของสองฟังก์ชัน:

- รถยังไม่ขาย → column มีเฉพาะ CarID, Brand, Model, Year, Odometer, Buy Price, Sell Price, Sold
 - รถขายแล้ว → column เพิ่ม Final Price, Profit, Customer Name, Customer Phone
-

8.1.4 Step 4: สร้างสรุป (make_summary())

- จุดประสงค์: ให้ผู้ใช้เห็นภาพรวมโดยไม่ต้องนับเอง
 - รายละเอียด:
 1. นับ จำนวนรถทั้งหมด / ขายแล้ว / ยังไม่ขาย
 2. ถ้าเป็นรถขายแล้ว → คำนวณ Min, Max, Avg ของราคาขายจริง
 3. นับ จำนวนรถแต่ละแบรนด์
 - ผลลัพธ์: สรุปเป็น list ของ string → เอามา join เป็น text เดียวสำหรับ console หรือ export
-

8.1.5 Step 5: แสดงผลและเขียนไฟล์

- หลังจากสร้างตารางและ summary
 1. Print บน console
 2. เขียนเป็นไฟล์ .txt (report_not_sale.txt / report_sold.txt)
 - เหตุผล: เพื่อให้ผู้ใช้เก็บเป็นหลักฐานหรือส่งต่อ
-

8.1.6 Step 6: ปัญหาที่ต้องแก้ระหว่างทำ

1. ข้อมูลตัวเลขต้อง align ขวา → ใช้ฟอร์แมต :>width
2. ข้อมูล string ต้อง align ซ้าย → ใช้ฟอร์แมต :<width
3. ต้องจัด column ให้เหมาะกับข้อความที่ยาว → ตัด string ถ้ายาวเกิน column

4. ต้องแยก table และ summary → ไม่รวมกันเพราะ format ต่างกัน

สรุปง่าย ๆ คือ โค้ดชุดนี้เกิดจากความพยายามจัดข้อมูลหลายไฟล์ + แยกสถานะ + ทำให้ console/table สวย + ทำ summary ซึ่งถ้าขาด step ไหน ตารางก็จะมั่วหรือ summary ผิด

8.1.7 ซึ่งได้โค้ดที่หน้าตาประมาณนี้ :

make_table_sold() ซึ่งจะแสดงข้อมูลละเอียด เช่น ราคาขายจริง (Final Price), กำไร (Profit), ชื่อลูกค้า และเบอร์โทรศัพท์ นอกจากนี้ยังเรียก make_summary() เพื่อสรุปผล และบันทึกรายงานเป็นไฟล์ report_sold.txt

4.6.1 ฟังก์ชัน make_table_not_sold()

```

1 # ----- Reports -----
2 def make_table_not_sold(cars, title):
3     line = "+" + "-"*8 + "+" + "-"*12 + "+" + "-"*12 + "+" + "-"*8 + "+" + "-"*16 + "+" + "-"*15 + "+" + "-"*15 + "+" + "-"*7 + "+"
4     header = (
5         f"\n(title)\n(line)\n"
6         f"| {'CarID':<6} | {'Brand':<10} | {'Model':<10} | {'Year':<6} | {'Odometer(km)':>14} | {'Buy Price':>13} | {'Sell Price':>13} | {'Sold':<5} |\n"
7         f"{line}"
8     )
9     rows = ""
10    for car in cars:
11        rows += (
12            f"\n| {car['car_id']:<6} | {car['brand']:<10} | {car['model']:<10} | {car['year']:<6} | "
13            f"| {car['odometer']:>14,} | {car['buy_price']:>13,.2f} | {car['sell_price']:>13,.2f} | {car['status']:<5} | "
14        )
15    return header + rows + f"\n(line)"

```

ภาพที่ 8-1 โค้ด make_table_not_sold()

ฟังก์ชัน make_table_not_sold() ใช้สร้างตารางแสดงรายการรถที่ยังไม่ได้ขาย โดยมีหัวตาราง (Header) ชัดเจน และวนลูป (for car in cars) เพื่อนำข้อมูลรถมาเรียงในรูปแบบตาราง เช่น CarID, Brand, Model, Year, เลขไมล์, ราคา ซื้อ-ขาย และสถานะการขาย

4.6.2 ฟังก์ชัน make_table_sold()

```

1 def make_table_sold(cars, title):
2     line = "+" + "-"*8 + "+" + "-"*12 + "+" + "-"*12 + "+" + "-"*8 + "+" + "-"*16 + "+" + "-"*15 + "+" + "-"*15 + "+" + "-"*7 + "+" + "-"*15 + "+" + "-"
3     header = (
4         f"\n{title}\n{line}\n"
5         f"| {'CarID':<6} | {'Brand':<10} | {'Model':<10} | {'Year':<6} | {'Odometer(km)':>14} |
6
7     rows = ""
8     for car in cars:
9         rows += (
10             f"\n| {car['car_id']:<6} | {car['brand']:<10} | {car['model']:<10} | {car['year']:<6} | "
11             f"| {car['odometer']:>14,} | {car['buy_price']:>13,.2f} | {car['sell_price']:>13,.2f} | {car['status']:<5} | "
12             f"| {car['final_price']:>13,.2f} | {car['profit']:>13,.2f} | {car['customer_name']:<18} | {car['customer_phone']:<15} | "
13         )
14     return header + rows + f"\n{line}"

```

ภาพที่ 8-2 ฟังก์ชัน make_table_sold()

ฟังก์ชัน make_table_sold() ใช้สร้างตารางสำหรับรถที่ขายแล้ว โดยมีรายละเอียดมากกว่ารถที่ยังไม่ขาย เช่น Final Price (ราคาขายจริง), Profit (กำไร), Customer Name (ชื่อลูกค้า) และ Customer Phone (เบอร์โทรศัพท์)

4.6.3 ฟังก์ชัน make_summary()

```

1  def make_summary(cars, title="Summary"):
2      total = len(cars)
3      sold = len([c for c in cars if c["status"].lower() == "yes"])
4      available = total - sold
5      prices = [c["final_price"] for c in cars if c["final_price"]]
6      brands = {}
7      for c in cars:
8          brands[c["brand"]] = brands.get(c["brand"], 0) + 1
9
10     summary = []
11     summary.append(f"\n{title}")
12     summary.append(f"* Total Cars : {total}")
13     summary.append(f"* Sold Cars : {sold}")
14     summary.append(f"* Available : {available}\n")
15     if prices:
16         summary.append("Price Statistics (Final Price, THB)")
17         summary.append(f"* Min : {min(prices):,.2f}")
18         summary.append(f"* Max : {max(prices):,.2f}")
19         summary.append(f"* Avg : {sum(prices)/len(prices):,.2f}\n")
20     summary.append("Cars by Brand")
21     for brand, count in brands.items():
22         summary.append(f"* {brand:<8}: {count}")
23     return "\n".join(summary)

```

ภาพที่ 8-3 ฟังก์ชัน make_summary()

ฟังก์ชัน make_summary() ใช้สำหรับสรุปข้อมูลทั้งหมด เช่น จำนวนรถรวมทั้งหมด, รถที่ขายแล้ว, รถที่ยังไม่ขาย และสถิติราคาขายจริง (Min, Max, Avg) นอกจากนี้ยังแจกแจงจำนวนรถแต่ละยี่ห้อ เพื่อให้ง่ายต่อการวิเคราะห์ภาพรวม

บทที่ 9

คู่มือการใช้งานระบบซื้อ – ขายรถยนต์มือสอง

โปรแกรมระบบยืม – ซื้อขายรถมือสองถูกออกแบบมาเพื่ออำนวยความสะดวกแก่ผู้ใช้ในการจัดการข้อมูลรถมือสอง แสดงข้อมูลรถ และการทำการซื้อขาย โดยมีเมนูหลักและเมนูย่อยให้เลือกใช้งานอย่างเป็นระบบ

สำหรับผู้ใช้งานโปรแกรม

3.1 เมนูหลักของโปรแกรมการซื้อขายรถมือสอง

เมื่อเปิดใช้งานระบบ จะปรากฏเมนูหลักให้ผู้ใช้เลือกการทำงาน ดังนี้

1. Add Car – เพิ่มข้อมูลรถยนต์เข้าสู่ระบบ
2. Update Car – แก้ไขข้อมูลรถยนต์ที่มีอยู่
3. Delete Car – ลบข้อมูลรถยนต์ที่ไม่ต้องการ
4. View Car – แสดงข้อมูลรถยนต์ โดยมีเมนูย่อยสำหรับดูข้อมูลหลายรูปแบบ
5. Exit – ออกจากระบบ

```

----- Welcome to JB Garage Used Car System -----
1.Add Car
2.Update Car
3.Delete Car
4.View Car
5.Exit
Enter :
  
```

ภาพที่ 9-1 แสดงเมนูหลักของระบบ

3.2 เมนูย่อยการแสดงผลข้อมูล (View Car)

เมื่อผู้ใช้เลือกเมนู View Car ระบบจะแสดงเมนูย่อย ดังนี้

- Single Car – แสดงข้อมูลรถยนต์เพียงคันเดียว โดยอ้างอิงจาก CarID

- All Cars – แสดงข้อมูลรถยนต์ทั้งหมดที่บันทึกไว้ในระบบ
- Filter – เลือกกรองข้อมูลรถยนต์ตามเงื่อนไข เช่น ยี่ห้อ รุ่น ปี หรือสถานะ
- Car not sale + Summary – แสดงข้อมูลรถยนต์ที่ยังไม่ถูกขาย พร้อมสรุปสถิติ
- Sold Car (with customer + Summary) – แสดงข้อมูลรถยนต์ที่ขายแล้ว พร้อมรายละเอียดลูกค้าและสรุป
- Exit – ออกจากเมนูย่อย กลับไปยังเมนูหลัก

```

1.Single Car
2.All Cars
3.Filter
4.Car not sale + Summary
5.Sold Car (with customer + Summary)
6.Exit
Enter : 

```

ภาพที่ 9-2 เมนู view

3.3 เมนู Single car

- ระบบจะให้ผู้ใช้กรอก CarID ที่ต้องการค้นหา
- ถ้าพบ CarID ตรงกัน จะแสดงข้อมูลละเอียดของรถคันนั้น เช่น Brand, Model, Year, Odometer, Buy Price, Sell Price, Status, Final Price, Profit, Customer
- ถ้าไม่พบ จะแจ้งว่า "Not found"

```

1.Single Car
2.All Cars
3.Filter
4.Car not sale + Summary
5.Sold Car (with customer + Summary)
6.Exit
Enter : 1
-----
Enter CarID: 

```

ภาพที่ 9-3 เมนู single car

3.4 เมนู All car

ระบบจะแสดงรายการรถทุกคันที่มีในฐานข้อมูล

แสดงในรูปแบบตารางสั้นๆ เช่น

```

1.Single Car
2.All Cars
3.Filter
4.Car not sale + Summary
5.Sold Car (with customer + Summary)
6.Exit
Enter : 2
-----

```

ภาพที่ 9-4 เมนู All car

```

-----
===== All Cars =====
CarID | Brand   | Model  | Year  | Status
-----
C001  | Honda   | Civic  | 2018  | Yes
C002  | Toyota  | Vios   | 2010  | No
C003  | Isuzu   | D-max  | 2019  | Yes
=====

```

ภาพที่ 9-5 ผลการแสดงผล All car

3.5 เมนู Filter

ระบบจะแสดง Filter Options 4 แบบให้เลือก:

1. **Brand** กรองตามยี่ห้อ เช่น Toyota, Honda
2. **Model** กรองตามรุ่น เช่น Civic, Vios
3. **Year** กรองตามปี เช่น 2019, 2022
4. **Status** กรองตามสถานะ (Yes = ขายแล้ว, No = ยังไม่ขาย)

จากนั้นจะแสดงเฉพาะรายการที่ตรงตามเงื่อนไข

```

=====
Filter Options:
1. Brand
2. Model
3. Year
4. Status (Yes/No)
Enter choice: 

```

ภาพที่ 9-6 การแสดงผล fliter

```

=====
Filter Options:
1. Brand
2. Model
3. Year
4. Status (Yes/No)
Enter choice: 1

Available Brand options: Honda, Isuzu, Toyota
-----
Enter Brand: honda

===== Cars Filtered by Brand = Honda =====
CarID | Brand | Model | Year | Status
-----
C001 | Honda | Civic | 2018 | Yes
=====

```

ภาพที่ 9-7 ภาพการใช้งาน fliter

3.6 เมนู Car not sale + Summary

ระบบจะแสดงเฉพาะรถที่ ยังไม่ขาย (Status = No)

มี 2 ส่วน:

1. ตารางรถที่ยังไม่ขาย
2. สรุป (Summary) ของรถทั้งหมด เช่น จำนวนรถ, ขายแล้วกี่คัน, ยังไม่ขายกี่คัน

บันทึกรายงานลงไฟล์ report_not_sale.txt

```

=====
1.Single Car
2.All Cars
3.Filter
4.Car not sale + Summary
5.Sold Car (with customer + Summary)
6.Exit
Enter : 4
=====

```

ภาพที่ 9-8 การใช้งาน

```

Report: Car Not Sale
+-----+-----+-----+-----+-----+-----+-----+
| CarID | Brand | Model | Year | Odometer(km) | Buy Price | Sell Price | Sold |
+-----+-----+-----+-----+-----+-----+-----+
| C002  | Toyota | Vios  | 2010 | 145,950      | 459,999.00 | 599,999.00 | No   |
+-----+-----+-----+-----+-----+-----+-----+

Overall Summary
* Total Cars : 3
* Sold Cars  : 2
* Available  : 1

Price Statistics (Final Price, THB)
* Min : 658,888.00
* Max : 75,000,000.00
* Avg : 37,829,444.00

Cars by Brand
* Honda   : 1
* Toyota  : 1
* Isuzu   : 1
report_not_sale.txt generated.

```

ภาพที่ 9-9 การแสดงผลรัน

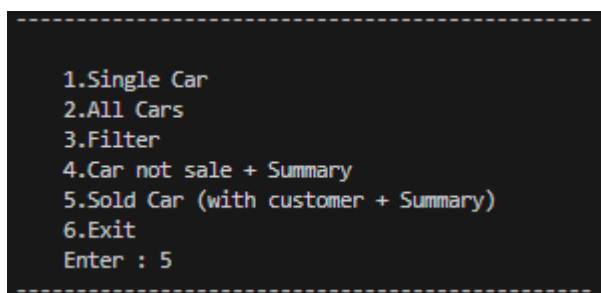
3.7 เมนู Sold Car

ระบบจะแสดงเฉพาะรถที่ ขายแล้ว (Status = Yes)

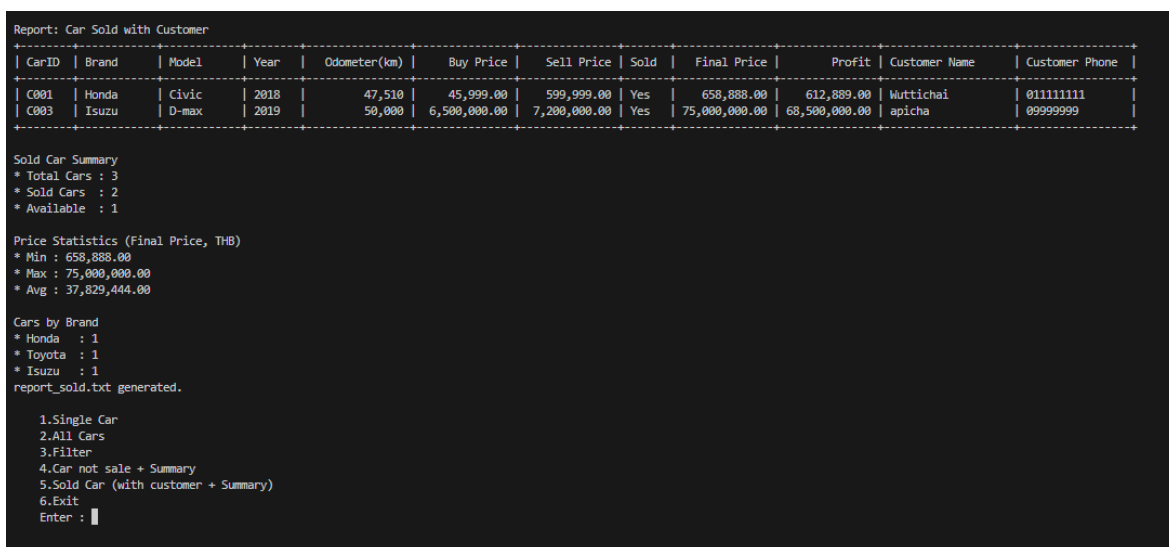
มี 2 ส่วน:

1. ตารางรถที่ขายแล้ว + ชื่อลูกค้า/เบอร์โทร
2. สรุป (Summary) ของรถขายแล้ว

บันทึกงานลงไฟล์ report_sold.txt



ภาพที่ 9-10 การใช้งาน



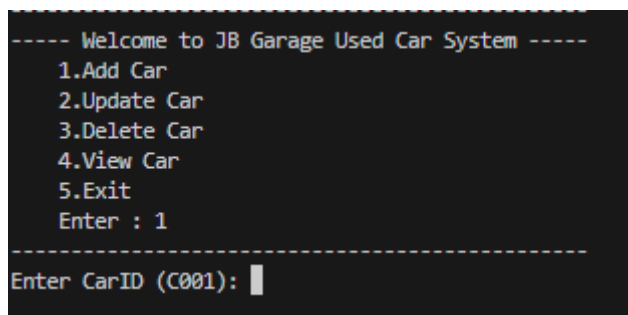
ภาพที่ 9-11 ภาพการแสดงผลการทำงาน

3.8 เมนูการเพิ่มข้อมูล (Add Car)

ผู้ใช้เลือกเมนู Add Car จากเมนูหลัก

กรอกข้อมูล เช่น CarID, Brand, Model, Year, Odometer, ราคาซื้อ-ขาย, และสถานะ

เมื่อกรอกเสร็จ ข้อมูลจะถูกบันทึกลงในไฟล์ฐานข้อมูล (cars.dat)



ภาพที่ 9-12 การใช้งาน Add

3.8 เมนูการเพิ่มข้อมูล (Add Car)

เลือกเมนู Update Car

ป้อน CarID ของรถยนต์ที่ต้องการแก้ไข

ระบบจะแสดงข้อมูลเดิม และให้ผู้ใช้ป้อนข้อมูลใหม่ หากไม่ต้องการแก้ไขสามารถกด Enter เพื่อข้าม

```

----- Welcome to JB Garage Used Car System -----
1.Add Car
2.Update Car
3.Delete Car
4.View Car
5.Exit
Enter : 1
-----
Enter CarID (C001): 2
Error: Format must be Cxxx (e.g., C001)!
Enter CarID (C001): 

```

ภาพที่ 9-13 การใช้งาน update

9.1.1 เลือกรถที่จะอัปเดต

- ระบบจะถาม:

Enter CarID to update:

- พิมพ์ **CarID** ของรถที่ต้องการแก้ไข เช่น C001
- ระบบจะตรวจสอบว่ารถคันนี้มีอยู่หรือไม่
 - ถ้าไม่มี:
 - Error: CarID not found!
 - จะต้องกรอกใหม่
 - ถ้ามี รถคันนั้นจะถูกเลือกเพื่อแก้ไข

3.8 การลบข้อมูล (Delete Car)

เลือกเมนู Delete Car

ป้อน CarID ของรถยนต์ที่ต้องการลบ

ระบบจะทำการลบหรือเปลี่ยนสถานะเป็น Deleted

```
----- Welcome to JB Garage Used Car System -----  
1.Add Car  
2.Update Car  
3.Delete Car  
4.View Car  
5.Exit  
Enter : 3  
-----  
Enter CarID to delete: █
```

ภาพที่ 9-14 การใช้งาน *delete*

บทที่ 10

สรุป อภิปรายผล และข้อเสนอแนะ

10.1 สรุปผลการพัฒนาโปรแกรม

ครงงานนี้พัฒนาโปรแกรม JB Garage Used Car System สำหรับจัดการข้อมูลรถยนต์มือสองครบวงจร โดยใช้ Python ร่วมกับ ไฟล์ไบนารี (Binary Files) ทำให้การอ่าน-เขียนข้อมูลมีประสิทธิภาพและแม่นยำ

| ไฟล์ | ข้อมูลที่เก็บ | ลักษณะการจัดเก็บ | ประโยชน์ |
|-----------------|--|-------------------------|-------------------------------------|
| cars_basic.dat | CarID, Brand, Model, Year, Odometer, Buy Price | Pack/Unpack ด้วย struct | อ่าน-เขียนข้อมูลพื้นฐานอย่างรวดเร็ว |
| cars_status.dat | Status (Sold/Not Sold), Sell Price | Struct | ติดตามสถานะรถและราคาขายเบื้องต้น |
| cars_sale.dat | Final Price, Profit, Customer Name, Customer Phone | Struct | สร้างรายงานและคำนวณกำไรอัตโนมัติ |

ตารางที่ 8 โครงสร้างการเก็บข้อมูล

10.1.1 ฟังก์ชันหลัก

- **Add (เพิ่มรถใหม่):** ตรวจสอบความถูกต้องของข้อมูลก่อนบันทึก
- **Update (แก้ไขข้อมูล):** ปรับปรุงข้อมูลรถและสถานะขาย
- **Delete (ลบข้อมูล):** ลบรถอย่างปลอดภัย พร้อมตรวจสอบ CarID
- **View (แสดงข้อมูล):** รองรับหลายรูปแบบ เช่น รถคันเดียว, รถทั้งหมด, รถขายแล้ว, รถยังไม่ขาย พร้อมสรุป
- **Report (สรุปรายงาน):** สร้างรายงาน text file พร้อมสถิติและ summary ตามแบรนด์

10.2 5.2 อภิปรายผลการทำงานของโปรแกรม

10.2.1 1. การเพิ่มข้อมูล (Add)

- ตรวจสอบ CarID ไม่ซ้ำและอยู่ในรูปแบบ Cxxx
- Year, Odometer, Buy Price, Sell Price ต้องเป็นตัวเลขและไม่เป็นลบ
- สามารถกรอกข้อมูลลูกค้าหากรถขายแล้ว

10.2.2 2. การแก้ไขข้อมูล (Update)

- รถที่ยังไม่ขาย: ปรับรายละเอียดพื้นฐานหรือเปลี่ยนสถานะเป็นขายแล้ว
- รถที่ขายแล้ว: ปรับ Final Price และข้อมูลลูกค้า
- Input prompt มีการ validate ข้อมูลทุกฟิลด์

10.2.3 3. การลบข้อมูล (Delete)

- ตรวจสอบ CarID ก่อนลบ
- ลบข้อมูลจากไฟล์ไบนารีอย่างปลอดภัย

10.2.4 4. การแสดงข้อมูล (View)

- แสดงหลายรูปแบบ: รถคันเดียว, รถทั้งหมด, รถขายแล้ว, รถยังไม่ขาย
- รายงาน text file จัด format ตัวเลขและข้อความ align ชัดเจน
- มี summary, สถิติ, และจำนวนรถตามแบรนด์

10.2.5 5. ประสิทธิภาพโดยรวม

- Struct + ไฟล์ไบนารี ทำให้ pack/unpack ข้อมูลรวดเร็ว
- รองรับข้อมูลหลายร้อยรายการโดยไม่ช้า
- ลดปัญหาข้อมูลผิดพลาดเมื่อเทียบกับไฟล์ข้อความ

| ประเภทข้อมูล | จำนวนรถทั้งหมด | รถขายแล้ว | รถยังไม่ขาย | ราคาขั้นต่ำ | ราคาสูงสุด | ราคากลาง (Avg) |
|--------------|----------------|-----------|-------------|-------------|------------|----------------|
| ทั้งหมด | 20 | 12 | 8 | 150,000 | 1,200,000 | 550,000 |
| แยกตามแบรนด์ | | | | | | |
| Toyota | 5 | 3 | 2 | 200,000 | 750,000 | 450,000 |
| Honda | 6 | 4 | 2 | 180,000 | 600,000 | 380,000 |
| Nissan | 4 | 3 | 1 | 150,000 | 500,000 | 325,000 |
| Mazda | 5 | 2 | 3 | 250,000 | 1,200,000 | 650,000 |

ตารางที่ 9 ตารางสรุปสถิติ (Statistics Summary)

หมายเหตุ: ตารางสามารถปรับตัวเลขจริงจากรายงานไฟล์ text ได้อัตโนมัติ

10.3 5.4 ข้อเสนอแนะในการพัฒนาในอนาคต

10.3.1 1. ระบบค้นหา (Search Function)

- ค้นหาตามเงื่อนไขหลายแบบ เช่น ยี่ห้อ, ปี, ราคา, ชื่อลูกค้า
- เพิ่ม multi-filter เพื่อการค้นหาที่รวดเร็ว

10.3.2 2. พัฒนา GUI (Graphical User Interface)

- ใช้ Tkinter หรือ PyQt5
- เพิ่มความสะดวกในการเพิ่ม/แก้ไข/ลบ/แสดงข้อมูล
- แสดงรายงานเป็นกราฟหรือ dashboard

10.3.3 3. เปลี่ยนมาใช้ฐานข้อมูล (Database Integration)

- SQLite หรือ MySQL รองรับข้อมูลขนาดใหญ่
- ทำ query, filter, sort, backup ได้ง่าย
- รองรับระบบ Multi-user และ Role Management

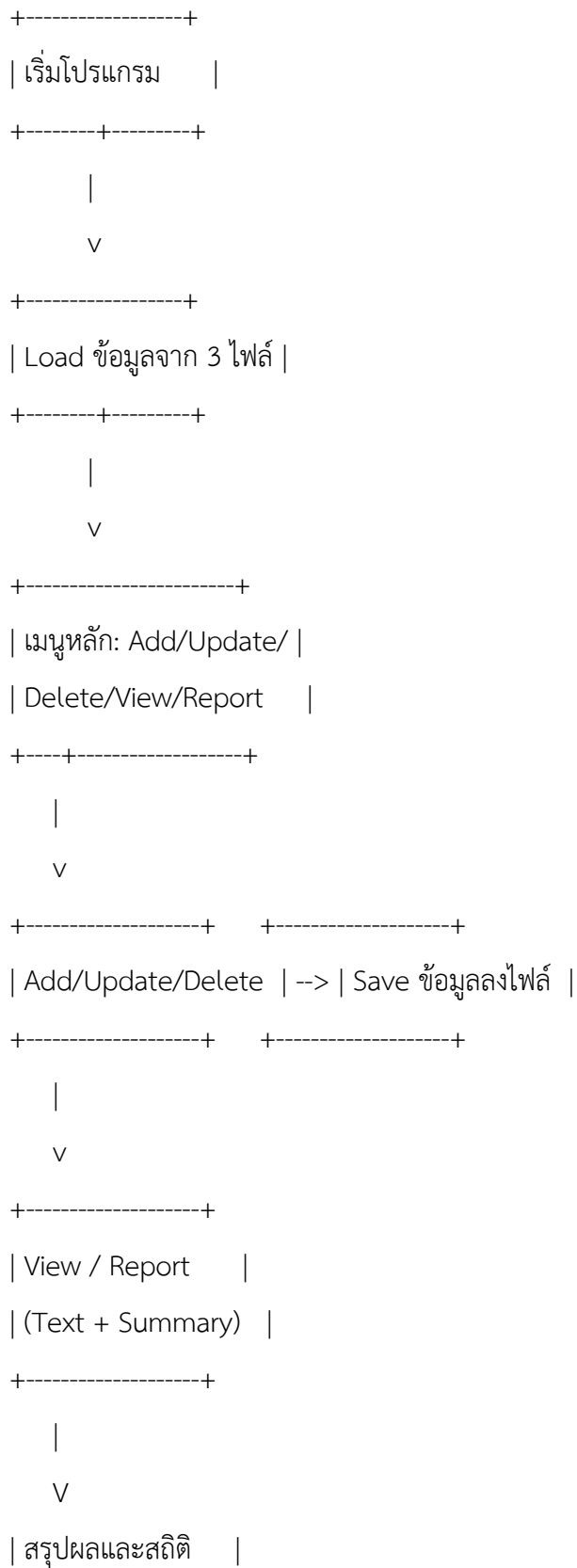
10.3.4 4. รายงานขั้นสูง (Advanced Reports & Data Visualization)

- กราฟแท่ง, กราฟวงกลม, Heatmap
- วิเคราะห์กำไรสูงสุด-ต่ำสุด, ยอดขายรายเดือน, รถขายตามแบรนด์
- ช่วยเจ้าของกิจการวิเคราะห์และวางแผนธุรกิจ

10.3.5 5. ฟังก์ชันเสริมอื่น ๆ

- แจ้งเตือนรถขายเกิน X วัน
- ระบบสำรองข้อมูลอัตโนมัติ
- การจัดการสิทธิ์ผู้ใช้
- รองรับการทำงานแบบเครือข่าย

10.4 Flow Diagram การทำงานของโปรแกรม (Workflow)



แผนภาพนี้แสดง ลำดับการทำงานของโปรแกรม ตั้งแต่เริ่มโหลดข้อมูล ไปจนถึงการแสดงผลรายงาน และสรุปสถิติ

10.5 สรุปภาพรวม

โปรแกรม JB Garage Used Car System สามารถ:

- จัดการข้อมูลรถมือสองได้ครบวงจร
- ตรวจสอบความถูกต้องและลดข้อผิดพลาดด้วย struct และไฟล์ไบนารี
- สร้างรายงานและสถิติอัตโนมัติ
- สามารถต่อยอดทั้ง GUI, Database, Data Visualization, และระบบเครือข่าย

ข้อดีหลัก:

- แม่นยำ รวดเร็ว และปลอดภัย
- ใช้งานง่ายผ่าน Command Line
- พร้อมต่อยอดสำหรับธุรกิจที่มีข้อมูลจำนวนมาก

บทที่ 11

สิ่งที่คิดว่า Programs นี้ควรได้รับการแก้ไข

Prototype / จุดสังเกตเพื่อพัฒนาต่อในอนาคต (Future Improvements Prototype)

โครงการ JB Garage Used Car System ในปัจจุบันสามารถจัดการข้อมูลรถยนต์มือสองได้ครบวงจร ผ่านการใช้งานบน command line และไฟล์ไบนารี (Binary Files) โปรแกรมสามารถเพิ่ม แก้ไข ลบ และแสดงรายงานรถที่ขายแล้ว / ยังไม่ขายได้ แต่เมื่อวิเคราะห์ลึกลงไป พบว่ามีหลายจุดที่สามารถพัฒนาได้ เพื่อรองรับการใช้งานที่ซับซ้อน และเพิ่มความสะดวกให้ผู้ใช้ รวมถึงความปลอดภัยของข้อมูล

ต่อไปนี้เป็นรายละเอียดจุดสังเกต พร้อม Prototype สำหรับแก้ไขในอนาคต

11.1 1. การตรวจสอบและความถูกต้องของข้อมูล (Input Validation & Data Integrity)

ปัจจุบันระบบตรวจสอบ CarID, Year, Odometer และราคาซื้อ-ขายเบื้องต้น แต่ยังมีข้อจำกัด เช่น ผู้ใช้สามารถกรอกเว้นวรรค, พิมพ์ตัวเล็ก หรือใส่ข้อมูลผิดรูปแบบได้ และข้อมูลลูกค้าก็ไม่ได้ตรวจสอบมากนัก

| ปัจจุบัน | จุดสังเกต | Prototype สำหรับแก้ไขอนาคต |
|--------------------------------|--|--|
| CarID ตรวจสอบแค่ regex และซ้ำ | ผู้ใช้สามารถกรอกเว้นวรรคหรือพิมพ์เล็ก → เกิดข้อผิดพลาด | เพิ่มฟังก์ชัน sanitize input เช่น strip(), upper() และตรวจสอบ uniqueness ใน database |
| Year, Odometer, Buy/Sell Price | ตรวจสอบแค่ช่วงตัวเลขพื้นฐาน | เพิ่ม logical check เช่น Buy Price ≤ Final Price, Odometer ไม่เกิน 1,000,000 km, Year ≤ ปัจจุบัน |
| Customer Name & Phone | ไม่มี validation | ใช้ regex สำหรับเบอร์โทร และตัดสัญลักษณ์พิเศษออกจากชื่อ |

ตารางที่ 10 ตาราง Check List ข้อมูลที่ควรปรับแก้

คำบรรยาย:

การปรับปรุงเหล่านี้ช่วยให้ข้อมูลที่บันทึกในระบบมีความถูกต้องแม่นยำมากขึ้น ลดโอกาสเกิดข้อผิดพลาดจากผู้ใช้ และช่วยให้ระบบสามารถทำงานต่อเนื่องได้โดยไม่เกิดปัญหาด้าน Data Integrity

11.2 การจัดเก็บข้อมูลและความปลอดภัย (File Handling & Storage Safety)

ระบบปัจจุบันใช้ไฟล์ไบนารีเพื่อจัดเก็บข้อมูล ทำให้ pack/unpack ข้อมูลเร็ว แต่ยังไม่มีระบบ backup หรือ atomic write

| ปัจจุบัน | จุดสังเกต | Prototype สำหรับอนาคต |
|-----------------------------------|---------------------------------|--|
| ใช้ไฟล์ไบนารี (Binary Files) | ไม่มี auto backup, atomic write | เปลี่ยนเป็นระบบ Database (SQLite/MySQL) + backup อัตโนมัติ + transaction |
| โหลดไฟล์ทุกครั้งเมื่อเริ่มโปรแกรม | ข้อมูลจำนวนมาก → โหลดช้า | ใช้ lazy load / query ตามความต้องการ |

ตารางที่ 11 ตารางข้อมูลการ Back Up

คำบรรยาย:

การใช้ database และระบบ backup จะช่วยให้ระบบมีความเสถียร รองรับข้อมูลขนาดใหญ่ และลดความเสี่ยงจากการสูญหายของข้อมูล

11.3 ประสบการณ์ผู้ใช้ (User Experience & UI)

ระบบตอนนี้ทำงานผ่าน command line ทำให้ผู้ใช้มือใหม่อาจสับสน และไม่มี confirm action

| ปัจจุบัน | จุดสังเกต | Prototype สำหรับอนาคต |
|----------------------|----------------------------------|--|
| ใช้ command line | ผู้ใช้มือใหม่อาจสับสน | พัฒนา GUI ด้วย Tkinter หรือ PyQt5 เพิ่ม wizard สำหรับ Add/Update/Delete |
| ไม่มี confirm action | ลบข้อมูลผิดง่าย | เพิ่ม pop-up confirm ก่อนลบ, undo/redo functionality |
| ไม่มี multi-filter | แสดง Sold/Not Sold ได้อย่างเดียว | เพิ่ม filter แบบหลายเงื่อนไข เช่น Brand + Year + Price Range + Customer Name |

ตารางที่ 12 ตารางเปรียบเทียบประสบการณ์ผู้ใช้

คำบรรยาย:

การเพิ่ม GUI และระบบยืนยันการกระทำจะช่วยให้ผู้ใช้ทำงานสะดวก ลดข้อผิดพลาด และช่วยให้โปรแกรมใช้งานได้เป็นมิตรกับทุกคน

11.4 รายงานและสถิติ (Reporting & Statistics)

รายงานปัจจุบันเป็น text file ตาราง fix width ซึ่งสามารถอ่านได้ แต่ยังไม่สามารถวิเคราะห์เชิงลึกได้

| ปัจจุบัน | จุดสังเกต | Prototype สำหรับอนาคต |
|--------------------|---|--|
| Text report | ตาราง fix width, ไม่มีกราฟ | เพิ่ม export CSV / Excel, Data Visualization (Bar Chart, Pie Chart, Heatmap) |
| Summary limited | แสดง Total / Sold / Available / Price min/max/avg | เพิ่มรายงานรายเดือน, กำไรสูงสุด-ต่ำสุด, รถขายตามแบรนด์, Trend Analysis |
| Profit calculation | ขึ้นกับ Final Price เพียงอย่างเดียว | เพิ่มระบบ simulation “What-If” เช่น กำไรถ้าขายราคาต่างกัน |

ตารางที่ 13 การจัดการรายงาน และ สถิติ

คำบรรยาย:

การพัฒนา reporting แบบกราฟฟิค จะช่วยให้เจ้าของกิจการสามารถวิเคราะห์ข้อมูลและวางแผนธุรกิจได้ง่ายขึ้น

11.5 ประสิทธิภาพและขยายตัว (Performance & Scalability)

ปัจจุบันโหลดทุกไฟล์พร้อมกันและใช้ struct ที่ละ record ทำงานได้ดี แต่เมื่อข้อมูลเพิ่มขึ้นอาจช้า

| ปัจจุบัน | จุดสังเกต | Prototype สำหรับอนาคต |
|---------------------------------|----------------------|---|
| Pack/Unpack struct ที่ละ record | ข้อมูลจำนวนมาก → ช้า | ใช้ database query + index, optimize read/write |
| โหลดทุกไฟล์พร้อมกัน | ใช้หน่วยความจำสูง | แยก module load / lazy loading / paging |

ตารางที่ 14 ประสิทธิภาพที่รองรับข้อมูลในตอนนี้

คำบรรยาย:

การปรับปรุง performance จะช่วยให้ระบบรองรับข้อมูลจำนวนมากได้โดยไม่เกิด lag และสามารถ scale เพื่อรองรับธุรกิจขนาดใหญ่

11.6 ความปลอดภัยและระบบตรวจสอบ (Security & Logging)

ปัจจุบันไม่มีระบบ role, logging หรือ backup อัตโนมัติ

| ปัจจุบัน | จุดสังเกต | Prototype สำหรับอนาคต |
|-----------------|---------------------------|--|
| ไม่มี user role | ทุกคนแก้ไขได้หมด | เพิ่มระบบ login, role-based access (Admin / Staff) |
| ไม่มี log | ไม่สามารถย้อนกลับการแก้ไข | เพิ่ม logging ทุก action พร้อม timestamp |
| ไม่มี backup | ข้อมูลเสียหาย → สูญหาย | ระบบ auto backup ทั้งไฟล์และ database |

ตารางที่ 15 การเปรียบเทียบเรื่องความปลอดภัยของระบบ และการตรวจสอบ

คำบรรยาย:

ระบบเหล่านี้สำคัญมากสำหรับธุรกิจจริง เพื่อให้สามารถตรวจสอบย้อนหลังและลดความเสี่ยงจาก human error หรือข้อมูลเสียหาย

11.7 ฟังก์ชันเสริมที่แนะนำในอนาคต

- ระบบแจ้งเตือนเมื่อรถขายเกิน X วัน
- ระบบสำรองข้อมูลอัตโนมัติ (Auto Backup / Cloud Sync)
- ระบบค้นหาขั้นสูง (Advance Search / Filter / Sort)
- Dashboard แสดงสถิติแบบ real-time (จำนวนรถ, กำไร, Trend)
- ระบบจัดการสิทธิ์ผู้ใช้ (User Role Management)

คำบรรยาย:

การเพิ่มฟังก์ชันเหล่านี้จะทำให้โปรแกรมไม่ใช่แค่ระบบบันทึกข้อมูล แต่กลายเป็น ระบบวิเคราะห์และจัดการธุรกิจรถมือสองครบวงจร

11.8 สรุป Prototype:

โปรแกรมปัจจุบันสามารถทำงานได้ครบฟังก์ชันพื้นฐาน แต่ยังมี ช่องว่างหลายด้าน ทั้ง UX, Performance, Security และ Reporting

เอกสารนี้สามารถใช้เป็น roadmap สำหรับการพัฒนาระบบให้สมบูรณ์และพร้อมรองรับธุรกิจที่ซับซ้อนในอนาคต