# ระบบจัดการข้อมูลร้านสะดวกซื้อ

# Convenience Store Management System


**นายตะวัน คำหนัน รหัส 6706022610128**

**นายธีรดนย์ จันพอน รหัส 6706022610438**

**นางอัมมาล อาแว รหัส  6706022610152**

**นายก้องเกียรติ กองอรรถ รหัส  6706022610225**


**โครงงานนี้เป็นส่วนหนึ่งของการศึกษาหลักสูตรวิศวกรรมศาสตรบัณฑิต**
**สาขาวิชาวิศวกรรมสารสนเทศและเครือข่าย ภาควิชาเทคโนโลยีสารสนเทศ**
**คณะเทคโนโลยีและการจัดการอุตสาหกรรม**
**มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ**
**ปีการศึกษา 2567**

# คำนำ

การจัดทำโครงงาน “ระบบจัดการข้อมูลร้านสะดวกซื้อ” นี้เป็นส่วนหนึ่งของวิชา COMPUTER PROGRAMMIMG ของหลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมสารสนเทศและเครือข่าย ภาควิชาเทคโนโลยีสารสนเทศคณะเทคโนโลยีและการจัดการอุตสาหกรรม มหาวิทยาลัยเทคโนโลยี พระจอมเกล้าพระนครเหนือ เพื่อให้นักศึกษาได้นำความรู้ที่เรียนมาทั้งหมดมาประยุกต์ใช้ในการ พัฒนาโปรแกรมที่สามารถทำงานได้จริง โดยเน้นการออกแบบและเขียนโปรแกรมในภาษา Python ซึ่งเป็นภาษาที่เรียนมาในวิชา COMPUTER PROGRAMMING โดยโครงงานนี้จะช่วย การคิดวิเคราะห์ และแก้ปัญหาทางเทคนิค เพื่อเตรียมความพร้อมในการประกอบอาชีพด้านวิศวกรรสารสนเทศและ เครือข่ายในอนาคต หากมีข้อผิดพลาดประการใด คณะผู้จัดทำต้องขออภัยไว้ ณ ที่นี้ด้วย

# สารบัญ

# สารบัญภาพ

# บทที่ 1
# บทนำ

## 1.1 วัตถุประสงค์ของโครงงาน

1.1.1 เพื่อพัฒนาระบบที่สามารถจัดการหนังสือได้อย่างมีประสิทธิภาพ

1.1.2 เพื่อฝึกฝนทักษะการเขียนโปรแกรมด้วย Python

1.1.3 เพื่อเรียนรู้การจัดการข้อมูลและไฟล์

1.1.4 เพื่อเรียนรู้การทำงานร่วมกันเป็นทีม

## 1.2 ขอบเขตของโครงงาน

1.2.1 ระบบจัดการข้อมูลสินค้าจะมีฟังก์ชันพื้นฐาน 8 ฟังก์ชัน เช่น 1. เมนูหลัก 2. การแสดง รายการหนังสือ 3. การเพิ่มหนังสือใหม่ 4. การอัปเดตสินค้า 5. การค้นหาสินค้า 6. การลบสินค้า 7. สร้างรายงาน 8. ออกจากโปรแกรม

1.2.2 ระบบจัดการข้อมูลหนังสือ ประกอบด้วย 5 ฟิลด์ ได้แก่ 1. record_id 2. name 3. price 4. category 5. stock_status

1.2.3 ระบบจัดการข้อมูลสินค้ามีการจัดเก็บข้อมูลหนังสือไว้ในไฟล์ text file

1.2.4 ระบบจัดการข้อมูลสินค้าจะมีเมนูเพื่อให้ผู้ใช้สามารถเลือกดำเนินการได้

## 1.3 ประโยชน์ที่ได้รับ

1.3.1 พัฒนาระบบที่สามารถจัดการหนังสือได้อย่างมีประสิทธิภาพ

1.3.2 พัฒนาทักษะการเขียนโปรแกรม

1.3.3 เรียนรู้การจัดการข้อมูลและไฟล์

1.3.4 เรียนรู้การทำงานร่วมกันเป็นทีม

## 1.4 เครื่องมือที่คาดว่าจะต้องใช้

1.4.1 ภาษา Python

1.4.2 Microsoft office

# บทที่ 2
# ระบบจัดการข้อมูลสินค้า

## 2.1 ฟิลด์ในระบบจัดการข้อมูลสินค้า

การจัดการข้อมูลสินค้าในระบบของคุณประกอบด้วย 5 ฟิลด์หลัก ซึ่งแต่ละฟิลด์มีรายละเอียดและความสำคัญดังนี้

### 2.1.1 **Product_id รหัสสินค้า (Identifier)**

record_id เป็นรหัสประจำสินค้าที่ใช้ในการระบุสินค้าแต่ละชิ้นอย่างเฉพาะเจาะจง ซึ่งมักจะเป็นตัวเลขที่ไม่ซ้ำกันในระบบ เช่น 1000, 1001, 1002 เป็นต้น รูปแบบของฟิลด์นี้เป็นประเภทข้อมูล **integer** (จำนวนเต็ม) มีความสำคัญในการค้นหาหรืออ้างอิงสินค้าในฐานข้อมูลได้ง่าย และช่วยป้องกันความสับสนระหว่างสินค้าที่อาจมีชื่อหรือรายละเอียดคล้ายกัน

### 2.1.2 **Name ชื่อสินค้า (Product Name)**

name คือชื่อของสินค้า ซึ่งสามารถระบุถึงชื่อเฉพาะของสินค้าชนิดนั้นๆ รูปแบบของฟิลด์นี้เป็นประเภทข้อมูล string (ข้อความ) เช่น "Water", "Apple", "Carrot" เป็นต้น ความสำคัญของฟิลด์คือชื่อสินค้านี้เป็นข้อมูลที่สำคัญสำหรับการค้นหาและการแสดงผลข้อมูลสินค้า โดยเฉพาะเมื่อผู้ใช้ต้องการค้นหาสินค้าโดยชื่อ ซึ่งช่วยในการจัดการสินค้าภายในระบบได้อย่างมีประสิทธิภาพและรวดเร็ว

### 2.1.3 **price ราคาของหนังสือ (Price)**

price คือ ราคาของหนังสือซึ่งแสดงถึงมูลค่าของหนังสือที่กำหนด ซึ่งรูปแบบของฟิลด์เป็นประเภทข้อมูล float (ทศนิยม) เช่น 150.50, 250.00 เป็นต้น ความสำคัญของฟิลด์นี้คือ ข้อมูลราคามีความสำคัญสำหรับผู้ใช้ในการตัดสินใจซื้อหนังสือ และช่วยให้ผู้จัดการระบบสามารถคำนวณยอดขายหรือกำไรได้ ข้อมูลราคา

ยังช่วยให้ผู้ใช้เปรียบเทียบราคาของหนังสือเล่มต่าง ๆ ได้อย่างสะดวก โดยเฉพาะในกรณีที่มีหนังสือหลายเล่มที่มีความคล้ายคลึงกัน

2.1.4 **category หมวดหมู่ของสินค้า (Category)**

category คือ หมวดหมู่ของสินค้าที่ระบุประเภทหรือกลุ่มของหนังสือ เช่น Drink (เครื่องดื่ม), Vegetable (ผัก), Fruit (ผลไม้) เป็นต้น รูปแบบของฟิลด์เป็นประเภทข้อมูล string (ข้อความ) ความสำคัญของฟิลด์นี้คือ การจัดหมวดหมู่หนังสือช่วยให้ผู้ใช้สามารถค้นหาหนังสือได้ง่ายขึ้น และช่วยในการจัดการข้อมูลหนังสือในระบบให้มีระเบียบเรียบร้อย นอกจากนี้ยังช่วยในการสร้างรายงานหรือสถิติที่เกี่ยวข้องกับประเภทหนังสือ

2.1.5 **stock_status สถานะสต็อกของหนังสือ (Stock Status)**

stock_status คือ สถานะของสต็อกหนังสือซึ่งบ่งบอกว่าสินค้าเล่มนั้นมีอยู่ในสต็อกหรือไม่ โดยมีค่าที่เป็นไปได้เช่น "In Stock" หรือ "Out Stock" รูปแบบของฟิลด์เป็นประเภทข้อมูล string (ข้อความ) ความสำคัญของฟิลด์นี้คือ สถานะสต็อกช่วยให้ผู้ใช้สามารถทราบได้ว่าหนังสือที่ต้องการมีพร้อมให้สั่งซื้อหรือไม่ ซึ่งจะช่วยให้ผู้ใช้สามารถตัดสินใจซื้อได้

2.1.6 **สรุปฟิลด์ในระบบจัดการข้อมูลสินค้า**

ฟิลด์ทั้ง 5 นี้ทำให้ระบบจัดการข้อมูลหนังสือสามารถทำงานได้อย่างมีประสิทธิภาพ โดยช่วยให้การจัดเก็บ การค้นหา การกรองข้อมูล และการแสดงผลข้อมูลเป็นไปได้อย่างรวดเร็วและมีประสิทธิภาพ นอกจากนี้ยังช่วยให้ผู้ใช้สามารถเข้าใจและเข้าถึงข้อมูลหนังสือได้ง่ายขึ้น ซึ่งจะส่งผลดีต่อประสบการณ์ของผู้ใช้ในการค้นหาและเลือกซื้อหนังสือ

## 2.2 ฟังก์ชันพื้นฐานในระบบจัดการข้อมูลร้านค้า

2.2.1 โครงสร้างและการทำงานของฟังก์ชัน main()

ฟังก์ชัน main() เป็นฟังก์ชันหลักของโปรแกรมระบบจัดการข้อมูลร้านค้า ทำหน้าที่หลักในการแสดงเมนูการจัดการข้อมูลและรับคำสั่งจากผู้ใช้เพื่อนำไปเรียกใช้ฟังก์ชันย่อยต่างๆ ตามที่เลือกจากเมนู ฟังก์ชันนี้ทำงานวนซ้ำในลูปจนกว่าผู้ใช้จะเลือกออกจากโปรแกรม โดยรายละเอียดของฟังก์ชันมีดังนี้:

2.2.1.1 โครงสร้างและการทำงานของฟังก์ชัน main()

แสดงเมนูหลัก: เมื่อโปรแกรมเริ่มต้น ฟังก์ชัน main() จะทำการแสดงเมนูหลักที่มีตัวเลือกต่างๆ ให้กับผู้ใช้ เช่น:

กด 1 เพื่อแสดงรายการสินค้า

กด 2 เพื่อเพิ่มสินค้าใหม่

กด 3 เพื่ออัปเดตข้อมูลสินค้า

กด 4 เพื่อค้นหาสินค้า

กด 5 เพื่อลบสินค้า

กด 6 เพื่อสร้างรายงาน

กด 7 เพื่อออกจากโปรแกรม

**การรับค่าการเลือกจากผู้ใช้**

ฟังก์ชัน input() จะถูกใช้ในการรับค่าจากผู้ใช้ โดยผู้ใช้ต้องใส่หมายเลขที่สอดคล้องกับฟังก์ชันที่ต้องการทำงาน

**การเรียกใช้ฟังก์ชันต่างๆ ตามการเลือกของผู้ใช้**

เมื่อผู้ใช้ใส่หมายเลขเลือกแล้ว โปรแกรมจะตรวจสอบค่าและเรียกใช้ฟังก์ชันที่สอดคล้องตามการเลือกนั้น หากผู้ใช้เลือกหมายเลขที่ไม่ถูกต้อง จะมีการแจ้งเตือนให้ผู้ใช้ทราบและขอให้ใส่หมายเลขใหม่

**การทำงานต่อเนื่อง**

หลังจากการทำงานของฟังก์ชันที่ถูกเลือกเสร็จสิ้น ฟังก์ชัน main() จะถูกเรียกซ้ำเพื่อแสดงเมนูอีกครั้ง ทำให้โปรแกรมทำงานต่อเนื่องจนกว่าผู้ใช้จะเลือกออกจากโปรแกรม

2.2.1.2 **โค้ดของฟังก์ชัน main()**

```python
def main():
    while True:
        print("\n" + "=" * 50)
        print("   Convenience Store Management Program")
        print("=" * 50)
        print("1. Add new data")
        print("2. Display all data")
        print("3. Retrieve specific data")
        print("4. Update data")
        print("5. Delete data")
        print("6. Create report")
        print("7. Exit program")

        choice = input("Please choose an option (1-7): ")

        if choice == '1':
            record_id = int(input("Please enter ID: "))
            name = input("Please enter name: ")
            price = float(input("Please enter price: "))
            category = input("Please enter category: ")
            stock_status = input("Please enter stock status (In Stock / Out of Stock): ")
            add_record(record_id, name, price, category, stock_status)

        elif choice == '2':
            display_records()

        elif choice == '3':
            search_value = input("Please enter ID or name to search: ")
            retrieve_records(search_value)

        elif choice == '4':
            try:
                record_id = int(input("Please enter the ID you want to update: "))
                new_name = input("Please enter new name: ")
                new_price = float(input("Please enter new price: "))
                new_category = input("Please enter new category: ")
                new_stock_status = input("Please enter new stock status (In Stock / Out Stock): ")
                update_record(record_id, new_name, new_price, new_category, new_stock_status)
            except ValueError:
```

ภาพที่ 2-1 โค้ดของฟังก์ชัน main()

```
def main():
    while True:
        print("\n" + "=" * 50)
        print("   Convenience Store Management Program")
        print("=" * 50)
        print("1. Add new data")
        print("2. Display all data")
        print("3. Retrieve specific data")
        print("4. Update data")
        print("5. Delete data")
        print("6. Create report")
        print("7. Exit program")

        choice = input("Please choose an option (1-7): ")

        if choice == '1':
            record_id = int(input("Please enter ID: "))
            name = input("Please enter name: ")
            price = float(input("Please enter price: "))
            category = input("Please enter category: ")
            stock_status = input("Please enter stock status (In Stock / Out of Stock): ")
            add_record(record_id, name, price, category, stock_status)

        elif choice == '2':
            display_records()

        elif choice == '3':
            search_value = input("Please enter ID or name to search: ")
            retrieve_records(search_value)

        elif choice == '4':
            try:
                record_id = int(input("Please enter the ID you want to update: "))
                new_name = input("Please enter new name: ")
                new_price = float(input("Please enter new price: "))
                new_category = input("Please enter new category: ")
                new_stock_status = input("Please enter new stock status (In Stock / Out Stock): ")
                update_record(record_id, new_name, new_price, new_category, new_stock_status)
            except ValueError:
                print("Invalid input. Please enter a valid ID.")

        elif choice == '5':
            try:
                record_id = int(input("Please enter the ID you want to delete: "))
                delete_record(record_id)
            except ValueError:
                print("Invalid input. Please enter a valid ID.")

        elif choice == '6':
            create_report()

        elif choice == '7':
            print("Closing the program")
            break

        else:
            print("Invalid option")

main()
```

ภาพที่ 2-2 โค้ดของฟังก์ชัน main() (2)

2.2.1.3 **แสดงหัวข้อของโปรแกรม**

เมื่อโปรแกรมเริ่มทำงาน ฟังก์ชัน main() จะพิมพ์หัวข้อและเมนูหลักของโปรแกรมด้วยสัญลักษณ์เครื่องหมายเท่ากับ (=) จำนวน 50 ตัว เพื่อสร้างความสวยงามในการแสดงผล จากนั้นจะแสดงชื่อโปรแกรมและเมนูการจัดการข้อมูล โดยการพิมพ์ข้อความถูกพิมพ์ด้วยคำสั่ง print()

**ชื่อโปรแกรม**: "Convenience Store Management"

**เส้นคั่น**: ใช้สัญลักษณ์ "=" เพื่อสร้างเส้นคั่น

**เมนูการจัดการ**: จะแสดงตัวเลือกต่าง ๆ ให้กับผู้ใช้

```
def main():
    while True:
        print("\n" + "=" * 50)
        print("   Convenience Store Management Program")
        print("=" * 50)
```

ภาพที่ 2-3 แสดงหัวข้อของฟังก์ชัน main()

2.2.1.4 **แสดงเมนูให้ผู้ใช้เลือก**

โปรแกรมจะเข้าสู่ลูป while True เพื่อแสดงเมนูซ้ำ ๆ และรับอินพุตจากผู้ใช้เรื่อย ๆจนกว่าจะเลือกเมนู "Exit" เพื่อออกจากโปรแกรม โดยลูป while True เป็นลูปที่ไม่เงื่อนไขสิ้นสุด ทำให้โปรแกรมทำงานต่อเนื่องไม่หยุด และรับอินพุตจากผู้ใช้ใช้คำสั่ง input() เพื่อให้ผู้ใช้เลือกหมายเลขเมนู จากนั้นแปลงข้อมูลที่ได้รับเป็นประเภทตัวเลขจำนวนเต็ม (int)

เมนูที่ผู้ใช้สามารถเลือกได้

Add new data: เพิ่มข้อมูลสินค้าใหม่

Display all data: แสดงข้อมูลสินค้าทั้งหมด

Retrieve specific data: ค้นหาสินค้าจากชื่อหรือรหัสสินค้า

Update data: แก้ไขข้อมูลสินค้าที่มีอยู่

Delete data: ลบข้อมูลสินค้า

Create data: สร้างรายงานของสินค้า

Exit program: ยืนยันการออกจากโปรแกรม

```python
def main():
    while True:
        print("\n" + "=" * 50)
        print("   Convenience Store Management Program")
        print("=" * 50)
        print("1. Add new data")
        print("2. Display all data")
        print("3. Retrieve specific data")
        print("4. Update data")
        print("5. Delete data")
        print("6. Create report")
        print("7. Exit program")

        choice = input("Please choose an option (1-7): ")
```

ภาพที่ 2-4 แสดงเมนูให้ผู้ใช้เลือกของฟังก์ชัน main()

2.2.1.5 **ตรวจสอบการเลือกของผู้ใช้**

เมื่อผู้ใช้เลือกหมายเลขจากเมนู ระบบจะตรวจสอบหมายเลขที่ผู้ใช้เลือก และเรียกใช้ฟังก์ชันที่เกี่ยวข้องโดยใช้โครงสร้างควบคุม if-elif-else ดังนี้:

ฟังก์ชันที่ถูกเรียกใช้:

add_record(): เรียกใช้เพื่อเพิ่มข้อมูลใหม่ลงในไฟล์.

display_records(): เรียกใช้เพื่อแสดงข้อมูลทั้งหมดในไฟล์.

retrieve_records(): เรียกใช้เพื่อค้นหาข้อมูลเฉพาะจากชื่อหรือรหัส.

update_record(): เรียกใช้เพื่ออัปเดตข้อมูลที่มีอยู่.

delete_record(): เรียกใช้เพื่อลบข้อมูลที่เลือก.

create_report(): เรียกใช้เพื่อสร้างรายงานจากข้อมูลทั้งหมด.

exit_program(): เรียกใช้เพื่อยืนยันการออกจากโปรแกรม.

หากผู้ใช้ป้อนหมายเลขที่ไม่อยู่ในช่วง 1 ถึง 7 โปรแกรมจะแสดงข้อความเตือนว่า "Invalid options" เพื่อแจ้งให้ผู้ใช้ทราบว่าเลขที่เลือกไม่ถูกต้อง

```
        else:
            print("Invalid option")

main()
```

ภาพที่ 2-5 การเลือกของผู้ใช้ฟังก์ชันหมายเลขที่ไม่อยู่ในช่วง 1-7 main()

2.2.1.6 **การวนลูปกลับสู่เมนูหลัก**

เนื่องจากลูป while True: จะทำงานต่อเนื่อง โปรแกรมจึงจะแสดงเมนูให้ผู้ใช้เลือกใหม่เรื่อย ๆ หลังจากทำงานเสร็จในแต่ละฟังก์ชัน เช่น เมื่อผู้ใช้แสดงรายการหนังสือหรือเพิ่มหนังสือเสร็จแล้ว โปรแกรมจะกลับมาที่เมนูหลักโดยอัตโนมัติ จนกว่าผู้ใช้จะเลือกเมนู "Exit program" (7) เพื่อออกจากโปรแกรม

2.2.1.7 **การออกจากโปรแกรม**

เมื่อผู้ใช้เลือกเมนู 7 โปรแกรมจะทำการ break main() และโปรแกรมหยุดทำงาน

```
    elif choice == 7:
        print("Closing the program")
        break
```

ภาพที่ 2-6 การออกจากโปรแกรมฟังก์ชัน main()

2.2.1.8 **สรุปการทำงานของฟังก์ชัน main()**

ฟังก์ชัน main() เป็นศูนย์กลางการทำงานของโปรแกรมทำหน้าที่แสดงเมนูและรับคำสั่งจากผู้ใช้มีการตรวจสอบอินพุตจากผู้ใช้และเรียกใช้ฟังก์ชันย่อยต่าง ๆ ตามเมนูที่เลือกและโปรแกรมจะวนลูปไปเรื่อย ๆ จนกว่าผู้ใช้จะเลือกออกจากโปรแกรม

2.2.1.9 **ผลลัพธ์ของฟังก์ชัน main()**

```
==========================================
   Convenience Store Management Program
==========================================
1. Add new data
2. Display all data
3. Retrieve specific data
4. Update data
5. Delete data
6. Create report
7. Exit program
Please choose an option (1-7):
```

ภาพที่ 2-7 ผลลัพธ์ของฟังก์ชัน main()

### 2.2.2 **โครงสร้างและการทำงานของฟังก์ชัน add_record()**

ฟังก์ชัน add_record() ทำหน้าที่ในการเพิ่มข้อมูลหนังสือใหม่ลงในไฟล์ data.bin โดยข้อมูลที่ถูกบันทึกจะมีรหัสหนังสือ ชื่อหนังสือ ราคา หมวดหมู่ และสถานะสต็อก

#### 2.2.2.1 **การทำงานของฟังก์ชัน**

ฟังก์ชันนี้อนุญาตให้ผู้ใช้เพิ่มข้อมูลผลิตภัณฑ์ใหม่ลงในระบบ โดยจะต้องป้อนค่าตามฟิลด์ที่กำหนดไว้ ได้แก่ record_id, name, price, category, และ stock_status ซึ่งมีรายละเอียดการทำงานดังนี้

**รับข้อมูลจากผู้ใช้**

ฟังก์ชัน add_record(record_id, name, price, category, stock_status) ถูกเรียกใช้เมื่อผู้ใช้ต้องการเพิ่มข้อมูลผลิตภัณฑ์ใหม่ โดยผู้ใช้จะต้องป้อนข้อมูลที่จำเป็นตามฟิลด์ที่กำหนด ซึ่งประกอบด้วย:

record_id: รหัสผลิตภัณฑ์ (ID)

name: ชื่อผลิตภัณฑ์

price: ราคาของผลิตภัณฑ์

category: หมวดหมู่ของผลิตภัณฑ์

stock_status: สถานะของสต็อก (In Stock / Out Stock)

```
if choice == 1:
    record_id = int(input("Please enter ID: "))
    name = input("Please enter name: ")
    price = float(input("Please enter price: "))
    category = input("Please enter category: ")
    stock_status = input("Please enter stock status (In Stock / Out of Stock): ")
    add_record(record_id, name, price, category, stock_status)
```

ภาพที่ 2-8 การใส่ข้อมูลสินค้าใหม่ main()

2.2.2.2 **เปิดไฟล์ในโหมดเพิ่มข้อมูล**

ฟังก์ชันจะเปิดไฟล์ data.bin ในโหมด 'ab' (append binary) เพื่อให้สามารถเขียนข้อมูลใหม่ลงในไฟล์โดยไม่ทำให้ข้อมูลเดิมที่มีอยู่ในไฟล์หายไป

```
add_record(record_id, name, price, category, stock_status):
with open(FILENAME, 'ab') as f:
```

ภาพที่ 2-9 การเปิดไฟล์ในโหมด 'ab' add_record()

2.2.2.3 **บีบอัดข้อมูล**

ข้อมูลที่ได้รับจากผู้ใช้จะถูกบีบอัดในรูปแบบไบนารี โดยใช้ฟังก์ชัน struct.pack() ซึ่งจะจัดเก็บข้อมูลตามรูปแบบที่กำหนดในตัวแปร RECORD_FORMAT ซึ่งประกอบด้วย:

I: รหัสผลิตภัณฑ์เป็นจำนวนเต็ม

20s: ชื่อผลิตภัณฑ์ (สูงสุด 20 ตัวอักษร)

f: ราคาของผลิตภัณฑ์เป็นจำนวนทศนิยม

20s: หมวดหมู่ของผลิตภัณฑ์ (สูงสุด 20 ตัวอักษร)

10s: สถานะของสต็อก (สูงสุด 10 ตัวอักษร)

```
FILENAME = "data.bin"
RECORD_FORMAT = 'I20sf20s10s'
RECORD_SIZE = struct.calcsize(RECORD_FORMAT)
```

ภาพที่ 2-10 การบีบอัดข้อมูลในรูปแบบ binary

2.2.2.4 **บันทึกข้อมูลลงในไฟล์**

ข้อมูลที่ถูกบีบอัดจะถูกเขียนลงในไฟล์ data.bin ซึ่งจะทำให้ข้อมูลผลิตภัณฑ์ใหม่ถูกเพิ่มเข้าไปในไฟล์ทันที

```
def add_record(record_id, name, price, category, stock_status):
    try:
        with open(FILENAME, 'ab') as f:
            f.write(struct.pack(RECORD_FORMAT, record_id, name.encode('utf-8'), price, category.encode('utf-8'), stock_status.encode('utf
```

ภาพที่ 2-11 ข้อมูลที่ใส่บีบอัดลงไฟล์ add_record()

2.2.2.5 **การแสดงผลของฟังก์ชั่น add_record()**

เมื่อข้อมูลถูกบันทึกลงในไฟล์เรียบร้อยแล้ว จะมีการแสดงข้อความยืนยันความสำเร็จและความผิดพลาด พร้อมกับรายละเอียดของสินค้าที่ถูกเพิ่มลงในระบบ

```
    try:
        with open(FILENAME, 'ab') as f:
            f.write(struct.pack(RECORD_FORMAT, record_id, name.encode('utf-8'), price, category.encode('utf-8'), stock_status.encode('utf

        print("\n" + "-" * 50)
        print(f"Record added successfully! ID: {record_id}, Name: {name}, Price: {price:.2f}, Category: {category}, Stock Status: {stock_
        print("-" * 50)

    except Exception as e:
        print("\n" + "-" * 50)
        print(f"Failed to add record: {str(e)}")
        print("-" * 50)
```

2.2.2.6 **ผลลัพธ์ของฟังก์ชั่น add_record()**

```
==================================================
   Convenience Store Management Program
==================================================
1. Add new data
2. Display all data
3. Retrieve specific data
4. Update data
5. Delete data
6. Create report
7. Exit program
Please choose an option (1-7): 1
Please enter ID: 10
Please enter name: Pencil
Please enter price: 1.00
Please enter category: Stationery
Please enter stock status (In Stock / Out Stock): Out Stock

--------------------------------------------------
Record added successfully! ID: 10, Name: Pencil, Price: 1.00, Category: Stationery, Stock Status: Out Stock
--------------------------------------------------
```

ภาพที่ 2-12 ผลลัพธ์ของฟังก์ชั่น add_record()

### 2.2.3 **โครงสร้างและการทำงานของฟังก์ชัน display_records()**

ฟังก์ชันนี้ทำหน้าที่ในการแสดงข้อมูลผลิตภัณฑ์ทั้งหมดที่มีอยู่ในไฟล์ data.bin โดยการอ่านข้อมูลจากไฟล์ในรูปแบบไบนารีและแสดงผลในรูปแบบที่เข้าใจง่าย ฟังก์ชันนี้มีรายละเอียดการทำงานดังนี้

#### 2.2.3.1 **ตรวจสอบการมีอยู่ของไฟล์**

ฟังก์ชันเริ่มต้นด้วยการตรวจสอบว่าไฟล์ data.bin มีอยู่ในระบบหรือไม่ หากไฟล์ไม่พบ ฟังก์ชันจะแสดงข้อความว่า "Data file not found" และหยุดการทำงาน

```
def display_records():
    if not os.path.exists(FILENAME):
        print("Data file not found")
        return
```

ภาพที่ 2-13 การตรวจสอบไฟล์ระบบ display_records()

#### 2.2.3.2 **แสดงหัวข้อข้อมูล**

หากไฟล์พบ ฟังก์ชันจะทำการพิมพ์เส้นขอบด้านบนและหัวข้อเพื่อเตรียมแสดงข้อมูล

```
    print("\n" + "-" * 50)
    print("Current Records:")
    print("-" * 50)
```

ภาพที่ 2-14 เตรียมแสดงข้อมูลหากพบไฟล์ display_record()

2.2.3.3 **เปิดไฟล์ในโหมดอ่านข้อมูล**

ฟังก์ชันจะเปิดไฟล์ data.bin ในโหมด 'rb' (read binary) เพื่อเริ่มการอ่านข้อมูลจากไฟล์

```
with open(FILENAME, 'rb') as f:
```

ภาพที่ 2-15 เปิดไฟล์ data.bin ในโหมด 'rb' display_record()

2.2.3.4 **อ่านข้อมูลจากไฟล์**

ฟังก์ชันจะใช้ลูป while เพื่ออ่านข้อมูลจากไฟล์เป็นชุด ๆ (records) ขนาดของแต่ละชุดจะถูกกำหนดโดย RECORD_SIZE ซึ่งคำนวณจากโครงสร้างข้อมูล

```
    while True:
        record = f.read(RECORD_SIZE)
        if not record:
            break
        record_id, name, price, category, stock_status = struct.unpack(RECORD_FORMAT, record)
```

ภาพที่ 2-16 ใช้ลูป while เพื่ออ่านข้อมูลจากไฟล์เป็นชุด display_record()

2.2.3.5 **แยกข้อมูลและแสดงผล**

สำหรับข้อมูลที่อ่านเข้ามาแต่ละชุด ฟังก์ชันจะทำการแยกข้อมูลออกเป็นฟิลด์ต่าง ๆ เช่น record_id, name, price, category, และ stock_status โดยใช้ struct.unpack(RECORD_FORMAT, record)

```
record_id, name, price, category, stock_status = struct.unpack(RECORD_FORMAT, record)
```

ภาพที่ 2-17 การแยกข้อมูลออกเป็นฟิลด์ต่าง ๆ display_record()

จากนั้น ฟังก์ชันจะแสดงข้อมูลแต่ละรายการในรูปแบบที่เป็นมิตรต่อผู้ใช้ โดยทำการแปลง name, category, และ stock_status จากไบนารีเป็นข้อความ (string) และตัดช่องว่างที่ไม่จำเป็นออก

```
print(f"ID: {record_id}, Name: {name.decode('utf-8').strip()}, Price: {price:.2f}, Category: {category.decode('utf-8').strip(
```

ภาพที่ 2-18 แสดงข้อมูลแต่ละรายการ display_record()

2.2.3.6 **แสดงเส้นขอบด้านล่าง**

หลังจากที่แสดงข้อมูลทั้งหมดแล้ว ฟังก์ชันจะพิมพ์เส้นขอบด้านล่างเพื่อให้รู้ว่าการแสดงข้อมูลสิ้นสุดลง

```
print("-" * 50)
```

ภาพที่ 2-19 พิมพ์เส้นขอบด้านล่าง display_record()

2.2.3.7 **การทำงานโดยรวมของฟังก์ชัน display_records()**

```
def display_records():
    if not os.path.exists(FILENAME):
        print("Data file not found")
        return

    print("\n" + "-" * 50)
    print("Current Records:")
    print("-" * 50)

    with open(FILENAME, 'rb') as f:
        while True:
            record = f.read(RECORD_SIZE)
            if not record:
                break
            record_id, name, price, category, stock_status = struct.unpack(RECORD_FORMAT, record)
            print(f"ID: {record_id}, Name: {name.decode('utf-8').strip()}, Price: {price:.2f}, Category: {category.decode('utf-8')
    print("-" * 50)
```

ภาพที่ 2-20 ทำงานโดยรวมของฟังก์ชัน display_records()

2.2.3.8 **ผลลัพธ์ของฟังก์ชัน display_record()**

```
================================================
   Convenience Store Management Program
================================================
1. Add new data
2. Display all data
3. Retrieve specific data
4. Update data
5. Delete data
6. Create report
7. Exit program
Please choose an option (1-7): 2

------------------------------------------------
Current Records:
------------------------------------------------
ID: 1, Name: Apple, Price: 1.50, Category: Fruit, Stock Status: In Stock
ID: 2, Name: Banana, Price: 0.75, Category: Fruit, Stock Status: In Stock
ID: 3, Name: Carrot, Price: 0.50, Category: Vegetable, Stock Status: Out Stock
ID: 4, Name: Broccoli, Price: 1.20, Category: Vegetable, Stock Status: In Stock
ID: 5, Name: Water Bottle, Price: 0.50, Category: Drink, Stock Status: In Stock
ID: 6, Name: Orange Juice, Price: 2.00, Category: Drink, Stock Status: Out Stock
ID: 7, Name: Bread, Price: 1.00, Category: Baked Goods, Stock Status: In Stock
ID: 8, Name: Eggs, Price: 2.50, Category: Dairy, Stock Status: In Stock
------------------------------------------------
```

ภาพที่ 2-21 ผลลัพธ์ของฟังก์ชัน display_record()

2.2.4 **โครงสร้างและการทำงานของฟังก์ชัน retrieve_records()**

ฟังก์ชันนี้มีวัตถุประสงค์เพื่อค้นหาและแสดงบันทึกข้อมูลจากไฟล์ที่ตรงตามค่าที่ผู้ใช้ระบุ โดยรายละเอียดการทำงานมีดังนี้:

2.2.4.1 **ตรวจสอบการมีอยู่ของไฟล์**

ฟังก์ชันเริ่มต้นด้วยการตรวจสอบว่าไฟล์ data.bin มีอยู่ในระบบหรือไม่ โดยใช้คำสั่ง os.path.exists(FILENAME) ถ้าไฟล์ไม่พบ ฟังก์ชันจะแสดงข้อความ "Data file not found" และใช้ return เพื่อหยุดการทำงาน

```python
def retrieve_records(search_value):
    if not os.path.exists(FILENAME):
        print("Data file not found")
        return
```

ภาพที่ 2-22 การตรวจสอบไฟล์ในระบบ retrieve_records()

2.2.4.2 **ตัวแปรสำหรับตรวจสอบการพบข้อมูล**

ฟังก์ชันจะสร้างตัวแปร found และตั้งค่าเป็น False เพื่อใช้ในการตรวจสอบว่ามีการพบข้อมูลที่ตรงตามที่ค้นหาหรือไม่

```
found = False
```

ภาพที่ 2-23 การตรวจสอบข้อมูลที่ตรง retrieve_records()

2.2.4.3 **เปิดไฟล์ในโหมดอ่านข้อมูล**

ฟังก์ชันจะเปิดไฟล์ data.bin ในโหมด 'rb' (read binary) โดยใช้คำสั่ง with open(FILENAME, 'rb') as f:

```
with open(FILENAME, 'rb') as f:
```

ภาพที่ 2-24 ฟังก์ชันจะเปิดไฟล์ในโหมด 'rb' retrieve_records()

2.2.4.4 **อ่านข้อมูลจากไฟล์**

ฟังก์ชันจะใช้ลูป while เพื่ออ่านข้อมูลจากไฟล์เป็นชุด ๆ ขนาดของแต่ละชุดจะถูกกำหนดโดย RECORD_SIZE

```
with open(FILENAME, 'rb') as f:
    while True:
        record = f.read(RECORD_SIZE)
        if not record:
            break
```

ภาพที่ 2-25 ใช้ลูป while เพื่ออ่านข้อมูลจากไฟล์เป็นชุด ๆ retrieve_records()

2.2.4.5 **แยกข้อมูลและตรวจสอบการจับคู่**

สำหรับข้อมูลที่อ่านเข้ามาแต่ละชุด ฟังก์ชันจะทำการแยกข้อมูลออกเป็นฟิลด์ต่าง ๆ เช่น record_id, name, price, category, และ stock_status โดยใช้ struct.unpack(RECORD_FORMAT, record)

```
        record_id, name, price, category, stock_status = struct.unpack(RECORD_FORMAT, record)
```

ภาพที่ 2-26 การแยกข้อมูลออกเป็นฟิลด์ต่าง ๆ retrieve_records()

ข้อมูล name, category, และ stock_status จะถูกแปลงจากไบนารีเป็นข้อความ (string) โดยใช้ decode('utf-8') และลบช่องว่างที่ไม่จำเป็นออกด้วย strip('\x00'):

```
        record_id, name, price, category, stock_status = struct.unpack(RECORD_FORMAT, record)
        name = name.decode('utf-8').strip('\x00')
        category = category.decode('utf-8').strip('\x00')
        stock_status = stock_status.decode('utf-8').strip('\x00')
```

ภาพที่ 2-27 แปลงจากไบนารีเป็นข้อความ retrieve_records()

2.2.4.6 **ค้นหาข้อมูลที่ตรงกัน**

ฟังก์ชันจะทำการตรวจสอบว่าค่าที่ป้อนเข้ามา (search_value) ตรงกับ record_id หรือ name หรือไม่ โดยใช้คำสั่ง if str(record_id) == search_value or name.lower() == search_value.lower()::

```
if str(record_id) == search_value or name.lower() == search_value.lower():
```

ภาพที่ 2-28 การตรวจสอบว่าค่าที่ป้อนเข้ามาตรงกับ record_id หรือ name

2.2.4.7 **แสดงผลเมื่อพบข้อมูลที่ตรงกัน**

หากพบข้อมูลที่ตรงกัน ฟังก์ชันจะแสดงผลบันทึกที่ตรงกันในรูปแบบที่เป็นมิตรต่อผู้ใช้ และตั้งค่า found เป็น True เพื่อบ่งบอกว่าพบข้อมูล:

```
print("\n" + "-" * 50)
print(f"Match found: ID: {record_id}, Name: {name}, Price: {price:.2f}, Category: {category}, Stock Status: {stock_status}")
print("-" * 50)
found = True
break
```

ภาพที่ 2-29 แสดงผลบันทึกที่ตรงกันในรูปแบบที่เป็นมิตรต่อผู้ใช้

2.2.4.8 **ตรวจสอบการไม่พบข้อมูล**

หลังจากลูปอ่านข้อมูลเสร็จสิ้น ฟังก์ชันจะตรวจสอบว่า found ยังคงเป็น False อยู่หรือไม่ หากใช่จะแสดงข้อความ "No matching records found." เพื่อบ่งบอกว่ามีข้อมูลที่ค้นหาไม่พบ

```
if not found:
    print("\nNo matching records found.")
```

ภาพที่ 2-30 แสดงข้อความว่ามีข้อมูลที่ค้นหาไม่พบ  recrieve_record()

### 2.2.4.9 **การทำงานโดยรวมของฟังก์ชัน retrieve_records()**

```
def retrieve_records(search_value):
    if not os.path.exists(FILENAME):
        print("Data file not found")
        return

    found = False
    with open(FILENAME, 'rb') as f:
        while True:
            record = f.read(RECORD_SIZE)
            if not record:
                break

            record_id, name, price, category, stock_status = struct.unpack(RECORD_FORMAT, record)
            name = name.decode('utf-8').strip('\x00')
            category = category.decode('utf-8').strip('\x00')
            stock_status = stock_status.decode('utf-8').strip('\x00')

            if str(record_id) == search_value or name.lower() == search_value.lower():
                print("\n" + "-" * 50)
                print(f"Match found: ID: {record_id}, Name: {name}, Price: {price:.2f}, Category: {category}, Stock Status: {stock_statu
                print("-" * 50)
                found = True
                break
```

ภาพที่ 2-31 การทำงานโดยรวมของฟังก์ชัน retrieve_records()

### 2.2.4.10 **ผลลัพธ์ของฟังก์ชัน retrieve_record()**

```
==================================================
   Convenience Store Management Program
==================================================
1. Add new data
2. Display all data
3. Retrieve specific data
4. Update data
5. Delete data
6. Create report
7. Exit program
Please choose an option (1-7): 3
Please enter ID or name to search: 1

--------------------------------------------------
Match found: ID: 1, Name: Apple, Price: 1.50, Category: Fruit, Stock Status: In Stock
--------------------------------------------------
```

ภาพที่ 2-32 ผลลัพธ์ของฟังก์ชัน retrieve_record()

2.2.5 **โครงสร้างและการทำงานของฟังก์ชัน update_record()**

2.2.5.1 **รับพารามิเตอร์ที่ต้องการสำหรับการอัปเดต**

ฟังก์ชันนี้รับค่าพารามิเตอร์ 5 ค่า record_id, new_name, new_price, new_category, และ new_stock_status ซึ่งเป็นค่าข้อมูลที่ต้องการอัปเดตสำหรับบันทึกข้อมูลที่มี record_id ตรงกัน

2.2.5.2 **ตัวแปรเก็บข้อมูลทั้งหมด**

ตัวแปร records ถูกใช้ในการเก็บข้อมูลทั้งหมดที่ถูกอ่านมาจากไฟล์เพื่อนำกลับไปเขียนใหม่ โดยจะเก็บทั้งบันทึกข้อมูลที่อัปเดตและบันทึกข้อมูลที่ไม่ต้องเปลี่ยนแปลง

```
def update_record(record_id, new_name, new_price, new_category, new_stock_status):
    records = []
```

ภาพที่ 2-33 การเก็บข้อมูลทั้งหมดที่ถูกอ่านมาจากไฟล์

2.2.5.3 **ตัวแปรเพื่อตรวจสอบการอัปเดต**

ตัวแปร updated ถูกกำหนดค่าเริ่มต้นเป็น False เพื่อใช้ในการตรวจสอบว่ามีการอัปเดตบันทึกข้อมูลในไฟล์หรือไม่

```
updated = False
```

ภาพที่ 2-34 การตรวจสอบว่ามีการอัปเดต

2.2.5.4 **เปิดไฟล์เพื่ออ่านข้อมูล**

ฟังก์ชันจะเปิดไฟล์ในโหมด 'rb' (read binary) เพื่อทำการอ่านข้อมูลทั้งหมดจากไฟล์และนำไปเก็บในตัวแปร records

```
with open(FILENAME, 'rb') as f:
```

ภาพที่ 2-35 เปิดไฟล์ในโหมด 'rb' update_record()

2.2.5.5 **ลูปอ่านข้อมูล**

ข้อมูลจะถูกอ่านเป็นบันทึก ๆ โดยแต่ละบันทึกมีขนาด RECORD_SIZE ฟังก์ชันจะใช้ struct.unpack(RECORD_FORMAT, record) เพื่อแปลงข้อมูลออกเป็นฟิลด์ต่าง ๆ ได้แก่ rec_id, name, price, category, และ stock_status:

```
with open(FILENAME, 'rb') as f:
    while True:
        record = f.read(RECORD_SIZE)
        if not record:
            break
        rec_id, name, price, category, stock_status = struct.unpack(RECORD_FORMAT, record)
```

ภาพที่ 2-36 ลูปอ่านข้อมูลเพื่อแปลงข้อมูลออกเป็นฟิลด์ updated_record()

2.2.5.6 **ตรวจสอบและอัปเดตข้อมูล**

ถ้า rec_id ของบันทึกตรงกับ record_id ที่ต้องการอัปเดต ฟังก์ชันจะเพิ่มข้อมูลที่อัปเดตลงในรายการ records โดยใช้ค่าพารามิเตอร์ใหม่ที่ผู้ใช้ป้อนเข้ามา และเปลี่ยนค่า updated เป็น True

```
if rec_id == record_id:
    records.append((record_id, new_name.encode('utf-8'), new_price, new_category.encode('utf-8'), new_stock_status.encode('utf-8')))
    updated = True
```

ภาพที่ 2-37 ถ้า rec_id ของบันทึกตรงกับ record_id ที่ต้องการอัปเดต

แต่ถ้า rec_id ไม่ตรงกัน บันทึกข้อมูลเดิมจะถูกเพิ่มเข้าไปใน records โดยไม่ทำการเปลี่ยนแปลง

```
else:
    records.append((rec_id, name, price, category, stock_status))
```

ภาพที่ 2-38 แต่ถ้า rec_id ไม่ตรงกัน

2.2.5.7 **เขียนข้อมูลที่อัปเดตกลับไปยังไฟล์:**

หลังจากอ่านข้อมูลเสร็จแล้ว ฟังก์ชันจะเปิดไฟล์อีกครั้งในโหมด 'wb' (write binary) เพื่อทำการเขียนข้อมูลทั้งหมดใน records กลับไปยังไฟล์

```
with open(FILENAME, 'wb') as f:
    for rec in records:
        f.write(struct.pack(RECORD_FORMAT, *rec))
```

ภาพที่ 2-39 เปิดไฟล์อีกครั้งในโหมด 'wb' update_record()

2.2.5.8 **แสดงผลการอัปเดต**

หากมีการอัปเดตข้อมูล (updated เป็น True) ฟังก์ชันจะแสดงข้อความว่า "Data has been updated!" มิฉะนั้นจะแสดงข้อความว่า "No matching record found." เพื่อบอกว่าข้อมูลที่ต้องการอัปเดตไม่พบ

```
print("\n" + "-" * 50)
if updated:
    print("Data has been updated!")
else:
    print("No matching record found.")
print("-" * 50)
```

ภาพที่ 2-40 หากมีการอัปเดตข้อมูลฟังก์ชันจะแสดงข้อความ update_record()

2.2.5.9 **การทำงานโดยรวมของฟังก์ชัน update_record()**

1. ฟังก์ชันจะเปิดไฟล์และอ่านข้อมูลทั้งหมดเก็บไว้ในรายการ records
2. ฟังก์ชันตรวจสอบว่า record_id ที่ต้องการอัปเดตตรงกับข้อมูลใดในไฟล์หรือไม่ หากตรงจะอัปเดตข้อมูลในบันทึกนั้นด้วยค่าพารามิเตอร์ใหม่ที่ส่งมา
3. ข้อมูลทั้งหมดถูกเขียนกลับลงในไฟล์อีกครั้ง
4. แสดงผลลัพธ์ว่ามีการอัปเดตสำเร็จหรือไม่

```
def update_record(record_id, new_name, new_price, new_category, new_stock_status):
    records = []
    updated = False
    with open(FILENAME, 'rb') as f:
        while True:
            record = f.read(RECORD_SIZE)
            if not record:
                break
            rec_id, name, price, category, stock_status = struct.unpack(RECORD_FORMAT, record)
            if rec_id == record_id:
                records.append((record_id, new_name.encode('utf-8'), new_price, new_category.encode('utf-8'), new_stock_status.encode('u
                updated = True
            else:
                records.append((rec_id, name, price, category, stock_status))

    with open(FILENAME, 'wb') as f:
        for rec in records:
            f.write(struct.pack(RECORD_FORMAT, *rec))

    print("\n" + "-" * 50)
    if updated:
        print("Data has been updated!")
    else:
        print("No matching record found.")
    print("-" * 50)
```

ภาพที่ 2-41 การทำงานโดยรวมของฟังก์ชัน update_record()

2.2.5.10 **ผลลัพธ์ของฟังก์ชัน display_record()**

```
==============================================
   Convenience Store Management Program
==============================================
1. Add new data
2. Display all data
3. Retrieve specific data
4. Update data
5. Delete data
6. Create report
7. Exit program
Please choose an option (1-7): 4
Please enter the ID you want to update: 1
Please enter new name: Apple
Please enter new price: 1.50
Please enter new category: Fruit
Please enter new stock status (In Stock / Out Stock): Out Stock

--------------------------------------------------
Data has been updated!
--------------------------------------------------
```

ภาพที่ 2-42 ผลลัพธ์ของฟังก์ชัน display_record()

### 2.2.6 **โครงสร้างและการทำงานของฟังก์ชัน delete_record()**

#### 2.2.6.1 **ประกาศตัวแปรสำหรับเก็บข้อมูลและสถานะการลบ**

ฟังก์ชันเริ่มต้นด้วยการประกาศตัวแปร records เพื่อเก็บข้อมูลทั้งหมดที่ถูกอ่านมาจากไฟล์ และตัวแปร deleted ที่มีค่าเริ่มต้นเป็น False เพื่อใช้ตรวจสอบว่ามีการลบข้อมูลหรือไม่

```
records = []
deleted = False
```

ภาพที่ 2-43 ประกาศตัวแปรสำหรับเก็บข้อมูลและสถานการณ์ลบ

#### 2.2.6.2 **เปิดไฟล์ในโหมดอ่าน**

ฟังก์ชันจะเปิดไฟล์ไบนารีในโหมดอ่าน ('rb') เพื่ออ่านข้อมูลทั้งหมดจากไฟล์

```
with open(FILENAME, 'rb') as f:
```

ภาพที่ 2-44 เปิดไฟล์ไบนารีในโหมดอ่าน delete_record()

#### 2.2.6.3 **ลูปอ่านข้อมูลจากไฟล์**

ฟังก์ชันจะใช้ลูปในการอ่านข้อมูลทีละบันทึก ขนาดของแต่ละบันทึกถูกกำหนดโดย RECORD_SIZE และข้อมูลในบันทึกจะถูกถอดรหัสโดยใช้ struct.unpack(RECORD_FORMAT, record) เพื่อแยกข้อมูลเป็นฟิลด์ต่าง ๆ ได้แก่ rec_id, name, price, category, และ stock_status

```
with open(FILENAME, 'rb') as f:
    while True:
        record = f.read(RECORD_SIZE)
        if not record:
            break
        rec_id, name, price, category, stock_status = struct.unpack(RECORD_FORMAT, record)
```

ภาพที่ 2-45 ใช้ลูปในการอ่านข้อมูลทีละบันทึก delelte_record()

2.2.6.4 **ตรวจสอบและลบข้อมูล**

หาก rec_id ของบันทึกตรงกับ record_id ที่ผู้ใช้ต้องการลบ ฟังก์ชันจะไม่เพิ่มบันทึกนั้นลงในรายการ records และตั้งค่า deleted เป็น True เพื่อระบุว่าพบและลบข้อมูลแล้ว

```
        if rec_id != record_id:
            records.append((rec_id, name, price, category, stock_status))
        else:
            deleted = True
```

ภาพที่ 2-46 ตรวจสอบและลบข้อมูล delete_record()

2.2.6.5 **เขียนข้อมูลที่เหลือกลับลงในไฟล์**

หลังจากอ่านข้อมูลทั้งหมดแล้ว ฟังก์ชันจะเปิดไฟล์ในโหมดเขียน ('wb') เพื่อเขียนข้อมูลทั้งหมดที่ไม่ถูกลบลงในไฟล์ใหม่

```
with open(FILENAME, 'wb') as f:
    for rec in records:
        f.write(struct.pack(RECORD_FORMAT, *rec))
```

ภาพที่ 2-47 เปิดไฟล์ในโหมดเขียน ('wb') delete_record()

2.2.6.6 **แสดงผลการลบข้อมูล**

หากมีการลบข้อมูล (ค่า deleted เป็น True) ฟังก์ชันจะแสดงข้อความว่า "Data has been deleted!" แต่หากไม่มีบันทึกที่ตรงกับ record_id จะมีการแสดงข้อความว่า "No matching record found." เพื่อระบุว่าไม่พบข้อมูลที่ต้องการลบ

```
print("\n" + "-" * 50)
if deleted:
    print("Data has been deleted!")
else:
    print("No matching record found.")
print("-" * 50)
```

ภาพที่ 2-48 แสดงผลการลบข้อมูล delete_record()

2.2.6.7 **การทำงานโดยรวมของฟังก์ชัน delete_record()**

1. ฟังก์ชันจะเปิดไฟล์และอ่านข้อมูลทั้งหมด

2. ตรวจสอบว่าบันทึกข้อมูลที่อ่านมาตรงกับ record_id หรือไม่ หากตรง ข้อมูลจะไม่ถูกเพิ่มไปในรายการ records ทำให้ข้อมูลนั้นถูกลบออกจากไฟล์
3. ข้อมูลที่เหลือจะถูกเขียนกลับลงในไฟล์
4. ฟังก์ชันแสดงข้อความเพื่อบอกผลลัพธ์ว่าข้อมูลถูกลบหรือไม่

```
def delete_record(record_id):
    records = []
    deleted = False
    with open(FILENAME, 'rb') as f:
        while True:
            record = f.read(RECORD_SIZE)
            if not record:
                break
            rec_id, name, price, category, stock_status = struct.unpack(RECORD_FORMAT, record)
            if rec_id != record_id:
                records.append((rec_id, name, price, category, stock_status))
            else:
                deleted = True

    with open(FILENAME, 'wb') as f:
        for rec in records:
            f.write(struct.pack(RECORD_FORMAT, *rec))

    print("\n" + "-" * 50)
    if deleted:
        print("Data has been deleted!")
    else:
        print("No matching record found.")
    print("-" * 50)
```

ภาพที่ 2-49 การทำงานโดยรวมของฟังก์ชัน delete_record()

### 2.2.6.8 **ผลลัพธ์การทำงานของฟังก์ชัน delete_record()**

```
==============================================
   Convenience Store Management Program
==============================================
1. Add new data
2. Display all data
3. Retrieve specific data
4. Update data
5. Delete data
6. Create report
7. Exit program
Please choose an option (1-7): 5
Please enter the ID you want to delete: 1

--------------------------------------------------
Data has been deleted!
--------------------------------------------------
```

ภาพที่ 2-50 ผลลัพธ์การทำงานของฟังก์ชัน delete_record()

### 2.2.7 **โครงสร้างและการทำงานของฟังก์ชัน create_report()**

#### 2.2.7.1 **ตรวจสอบการมีอยู่ของไฟล์**

ฟังก์ชันเริ่มต้นด้วยการตรวจสอบว่าไฟล์ข้อมูลมีอยู่หรือไม่ ถ้าไม่มีไฟล์อยู่ ฟังก์ชันจะพิมพ์ข้อความแจ้งเตือนและหยุดการทำงาน

```
def create_report():
    if not os.path.exists(FILENAME):
        print("Data file not found")
        return
```

ภาพที่ 2-51 ตรวจสอบการมีอยู่ของไฟล์ create_report()

#### 2.2.7.2 **สร้างตัวแปรสำหรับรายงาน**

ประกาศตัวแปร report_lines เพื่อใช้เก็บข้อมูลจากไฟล์เป็นบรรทัดต่าง ๆ ซึ่งจะถูกใช้ในการเขียนลงไฟล์ report.txt ภายหลัง

```
report_lines = []
```

ภาพที่ 2-52 สร้างตัวแปรสำหรับรายงาน create_report()

#### 2.2.7.3 **เปิดไฟล์ในโหมดอ่าน**

ฟังก์ชันจะเปิดไฟล์ไบนารีในโหมดอ่าน ('rb') เพื่ออ่านข้อมูลทีละบันทึก โดยใช้ลูปเพื่ออ่านบันทึกทั้งหมดในไฟล์

```
with open(FILENAME, 'rb') as f:
```

ภาพที่ 2-53 เปิดไฟล์ในโหมดอ่าน create_report()

#### 2.2.7.4 **อ่านและจัดรูปแบบข้อมูล**

ข้อมูลที่อ่านมาจากไฟล์จะถูกจัดรูปแบบเป็นบรรทัดที่มีฟอร์แมตอ่านง่าย โดยใช้ struct.unpack() เพื่อถอดข้อมูลเป็นฟิลด์ต่าง ๆ เช่น record_id, name, price, category, และ stock_status ข้อมูลเหล่านี้จะถูกแปลงเป็นข้อความและถูกเพิ่มลงใน report_lines

```
while True:
    record = f.read(RECORD_SIZE)
    if not record:
        break
    record_id, name, price, category, stock_status = struct.unpack(RECORD_FORMAT, record)
    name = name.decode('utf-8').strip('\x00')
    category = category.decode('utf-8').strip('\x00')
    stock_status = stock_status.decode('utf-8').strip('\x00')
    line = f"ID: {record_id}, Name: {name}, Price: {price:.2f}, Category: {category}, Stock Status: {stock_status}"
    report_lines.append(line)
```

ภาพที่ 2-54 อ่านและจัดรูปแบบข้อมูล create_report()

2.2.7.5 **เขียนรายงานลงในไฟล์**

ฟังก์ชันจะเปิดไฟล์ report.txt ในโหมดเขียน ('w') และเขียนบรรทัดทั้งหมดที่ถูกเก็บไว้ใน report_lines ลงในไฟล์ โดยใช้การเชื่อมบรรทัดด้วย \n

```
with open("report.txt", 'w', encoding='utf-8') as report_file:
    report_file.write("\n".join(report_lines))
```

ภาพที่ 2-55 เขียนรายงานลงในไฟล์ create_report()

2.2.7.6 **แสดงผลลัพธ์การสร้างรายงาน**

หลังจากสร้างรายงานเสร็จ ฟังก์ชันจะพิมพ์ข้อความแสดงความสำเร็จว่ารายงานได้ถูกบันทึกลงในไฟล์ report.txt แล้ว

```
print("\n" + "-" * 50)
print("Report saved to report.txt")
print("-" * 50)
```

ภาพที่ 2-56 แสดงผลลัพธ์การสร้างรายงาน create_report()

2.2.7.7 **การทำงานโดยรวมของฟังก์ชัน create_report()**

1. ตรวจสอบว่ามีไฟล์ข้อมูลอยู่หรือไม่ หากไม่พบไฟล์จะแสดงข้อความและหยุดการทำงาน

2. เปิดไฟล์ข้อมูลไบนารีเพื่ออ่านข้อมูลแต่ละบันทึก

3. จัดรูปแบบข้อมูลให้อยู่ในรูปแบบบรรทัดที่อ่านง่ายและเพิ่มลงในรายการ report_lines

4. บันทึกรายงานที่ประกอบด้วยข้อมูลทั้งหมดลงในไฟล์ report.txt

5. แสดงผลลัพธ์ให้ผู้ใช้ทราบว่ารายงานถูกบันทึกสำเร็จ

```python
def create_report():
    if not os.path.exists(FILENAME):
        print("Data file not found")
        return

    report_lines = []
    with open(FILENAME, 'rb') as f:
        while True:
            record = f.read(RECORD_SIZE)
            if not record:
                break
            record_id, name, price, category, stock_status = struct.unpack(RECORD_FORMAT, record)
            name = name.decode('utf-8').strip('\x00')
            category = category.decode('utf-8').strip('\x00')
            stock_status = stock_status.decode('utf-8').strip('\x00')
            line = f"ID: {record_id}, Name: {name}, Price: {price:.2f}, Category: {category}, Stock Status: {stock_status}"
            report_lines.append(line)

    with open("report.txt", 'w', encoding='utf-8') as report_file:
        report_file.write("\n".join(report_lines))

    print("\n" + "-" * 50)
    print("Report saved to report.txt")
    print("-" * 50)
```

ภาพที่ 2-57 การทำงานโดยรวมของฟังก์ชัน create_report()

2.2.7.8 **ผลลัพธ์ของฟังก์ชัน create_report()**

ฟังก์ชันจะสร้างรายงานในรูปแบบข้อความที่อ่านง่าย โดยแต่ละรายการจะถูกแสดงในบรรทัดเดียว และข้อมูลแต่ละฟิลด์จะถูกจัดเรียงอย่างชัดเจน

```
ID: 1, Name: Apple, Price: 1.50, Category: Fruit, Stock Status: In Stock
ID: 2, Name: Banana, Price: 0.75, Category: Fruit, Stock Status: In Stock
ID: 3, Name: Carrot, Price: 0.50, Category: Vegetable, Stock Status: Out Stock
ID: 4, Name: Broccoli, Price: 1.20, Category: Vegetable, Stock Status: In Stock
ID: 5, Name: Water Bottle, Price: 0.50, Category: Drink, Stock Status: In Stock
ID: 6, Name: Orange Juice, Price: 2.00, Category: Drink, Stock Status: Out Stock
ID: 7, Name: Bread, Price: 1.00, Category: Baked Goods, Stock Status: In Stock
ID: 8, Name: Eggs, Price: 2.50, Category: Dairy, Stock Status: In Stock
```

ภาพที่ 2-58 ผลลัพธ์ของฟังก์ชัน create_report()