
Instructions

Develop an object-oriented Python application that uses either **PyQt6** or **Tkinter** for its graphical user interface (GUI). The program must include a **minimum of three windows/forms**, including a main menu, and implement **full CRUD (Create, Read, Update, Delete)** operations on a **MySQL database**. The database should consist of **at least two related tables** using **Structured Query Language (SQL)**.

Application Name

Birthday Tracker: A Birthday Management and Reminder System

System Description

The **Birthday Tracker Application** is designed to help users organize and manage their contacts along with personalized gift ideas and important date reminders such as birthdays. The system centralizes information about people, their gift preferences, and upcoming birthdays in an easy-to-use graphical interface.

This application is intended for **individuals who want to streamline their gift planning process and never miss important occasions**. Users can add, edit, or delete contact details and gift ideas, and receive timely reminders about upcoming birthdays, allowing for better preparation and thoughtful gift-giving.

Core Windows (excluding main window)

- **Manage People**: Add and organize contacts with their details.
- **Gift Ideas**: Maintain a list of gift suggestions linked to contacts.
- **Reminders**: Track and display upcoming birthdays to help users plan ahead.

Core Python Files

- **main.py**: The application's entry point; handles window navigation and general startup logic.
- **db.py**: Contains all SQLite database operations and queries.
- **reminders.py**: Handles the logic and functionality specific to reminders.

Python Modules Used

This project utilizes a mix of Python standard libraries, third-party packages, and PyQt6 modules to support GUI functionality, data handling, and notifications.

Standard Library Modules

- **sqlite3**: Provides a lightweight, built-in interface for working with SQLite databases.
- **datetime**: Used for managing and formatting dates (especially for reminders and birthdays).
- **sys**: Enables control over the Python runtime environment, especially for application startup.

Third-Party Modules

- **plyer.notification**: Enables cross-platform desktop notifications for upcoming birthdays.

PyQt6 Modules

1. **PyQt6.QtWidgets**: Provides core UI components.
    - **QMainWindow**: Base class for the main application window.
    - **Qapplication**: Manages application-level settings and the event loop.
    - **Qwidget**: Base class for all UI windows (e.g., AddPersonWindow, EditPersonWindow).
    - **QmessageBox**: Displays dialog boxes for warnings, confirmation, and information.
    - **Qlabel**: Used in UI forms to show text (loaded via .ui file).
    - **Qframe**: Structural UI element (used in .ui files for grouping or styling).

2. **PyQt6.QtGui**: Handles UI models and items.
    - **QstandardItemModel**: A data model to hold and manage table data, used in QTableView,
    - **QStandardItem**: Represents individual cells/items in a QStandardItemModel.

3. **PyQt6.QtCore**: Provides core non-GUI functionality
    - **Qt**: Contains enums for properties (e.g., WindowModality.ApplicationModal).
    - **QData**: Handles date input/output (e.g., for birth dates).

4. **PyQt6.uic**: Allows loading of .ui files designed in Qt Designer.
    - **loadUi**: loads and applies a .ui files to a class instance (used for all windows).

Why I Used SQLite Instead of MySQL

At the beginning of the project, I attempted to use **MySQL** as the database backend, thinking it would be a robust option for managing relational data. However, this choice quickly became a major obstacle in development.

Technical Roadblocks with MySQL

- **Windows Closed Without Error**: When I opened windows in the app that performed SQL operations, they unexpectedly closed without displaying any error messages, making debugging very difficult.
- **Missing MySQL Driver**: I later discovered that my Qt environment didn't include the required QMYSQL driver needed to connect GUI actions to MySQL database operations. Setting this up was far from simple and required additional configurations that were not straightforward.
- **Qt Installer Problems**: In an attempt to reinstall or fix my Qt setup, I encountered multiple download issues using the Qt installer. Certain essential files simply wouldn't download, halting my progress further.
- **Disrupted Workflow**: These issues completely diverted my focus. Instead of designing the UI or implementing application logic, I spent most of my time troubleshooting environment problems.
- **Risk of Breaking the Environment**: With each attempted fix, I grew increasingly worried that I might corrupt my Python environment or damage important files, making things worse instead of better.
- **Frustration and Burnout**: The technical difficulties were frustrating and physically tiring, and I realized I was sacrificing productivity and well-being just trying to make MySQL work.

To get back on track and prioritize functionality, I made the decision to switch to **SQLite**. This switch allowed me to complete the core functionalities and operations without compromising the database design or user experience.

List of Tables

Below is a detailed description of each table in the database, including their columns, data types, and constraints.

1. **people**
   Stores basic personal information for each individual.

Column Name	Data Type	Constraints	Description
person_id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique identifier for each person
first_name	TEXT	NOT NULL	Person's first name
last_name	TEXT	NOT NULL	Person's last name
birth_date	TEXT	NOT NULL	Person's date of birth (YYYY-MM-DD)
email	TEXT		Person's email address
phone_number	TEXT		Person's phone number
notes	TEXT		Additional notes

2. **gift_ideas**
   Stores reminder entries associated with individuals, such as birthdays or other important dates.

Column Name	Data Type	Constraints	Description
gift_id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique identifier for each gift idea
person_id	INTEGER	FOREIGN KEY REFERENCES people(person_id) ON DELETE CASCADE	Links gift to a person
gift_name	TEXT	NOT NULL	Name or title of the gift idea
description	TEXT		Detailed description of the gift
place	TEXT		Where to buy or find the gift
link	TEXT		Online store link
price	DECIMAL (10, 2)		Estimated price
notes	TEXT		Additional notes

3. **reminders**
   Stores reminder entries associated with individuals, such as birthdays or other important dates.

Column Name	Data Type	Constraints	Description
reminder_id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique identifier for each reminder
person_id	INTEGER	NOT NULL, FOREIGN KEY REFERENCES people(person_id) ON DELETE CASCADE	Links reminder to a person

SQL Scripts

This section outlines all SQL statements used to create the database structure and to perform key operations such as inserting, updating, deleting, and retrieving records.

1. **Create Database and Tables**

```
CREATE TABLE IF NOT EXISTS people (
    person_id INTEGER PRIMARY KEY AUTOINCREMENT,
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    birth_date TEXT NOT NULL,
    email TEXT,
    phone_number TEXT,
    notes TEXT
);

CREATE TABLE IF NOT EXISTS gift_ideas (
    gift_id INTEGER PRIMARY KEY AUTOINCREMENT,
    person_id INTEGER,
    gift_name TEXT NOT NULL,
    description TEXT,
       place TEXT,
    link TEXT,
    price DECIMAL(10, 2),
       notes TEXT,
    FOREIGN KEY (person_id) REFERENCES people(person_id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS reminders (
    reminder_id INTEGER PRIMARY KEY AUTOINCREMENT,
    person_id INTEGER NOT NULL,
    FOREIGN KEY (person_id) REFERENCES people(person_id) ON DELETE CASCADE
);
```

2. **People Table Operations**

   a. **Insert Person**

```
INSERT INTO people (first_name, last_name, birth_date, email, phone_number, notes)
VALUES (?, ?, ?, ?, ?, ?);
```

   b. **Edit Person**

```
UPDATE people SET first_name = ?, last_name = ?, birth_date = ?, email = ?,
phone_number = ?, notes = ? WHERE person_id = ?;
```

   c. **Delete Person**

```
DELETE FROM people WHERE person_id = ?;
```

   d. **Get People List**

```
SELECT person_id, first_name, last_name, birth_date, email, phone_number, notes FROM
people;
```

3. **Gift Ideas Table Operations**

   a. **Insert Gift Idea**

```
INSERT INTO gift_ideas (person_id, gift_name, description, place, link, price, notes)
VALUES (?, ?, ?, ?, ?, ?, ?);
```

   b. **Edit Gift Idea**

```
UPDATE gift_ideas SET person_id = ?, gift_name = ?, description = ?, place = ?, link
= ?, price = ?, notes = ? WHERE gift_id = ?;
```

   c. **Delete Gift Idea**

```
DELETE FROM gift_ideas WHERE gift_id = ?;
```

   d. **Get Gift Ideas**

```
SELECT
    g.gift_id,
    p.first_name || ' ' || p.last_name AS full_name,
```

```
        g.gift_name,
        g.description,
        g.place,
        g.link,
        g.price,
        g.notes
    FROM gift_ideas g
    JOIN people p ON g.person_id = p.person_id;
```

4. **Reminder Table Operations**

    a. **Automatically Add Reminder for New Person**

    ```
    INSERT INTO reminders (person_id) VALUES (?);
    ```

    b. **Get People with Reminders**

    ```
    SELECT p.person_id, p.first_name, p.last_name, p.birth_date
    FROM people p
    JOIN reminders r ON p.person_id = r.person_id;
    ```

Complete Python Source Codes

A. **main.py**

```
from PyQt6.QtWidgets import QMainWindow, QApplication, QWidget, QMessageBox, QHBoxLayout, QLabel, QCheckBox,
QLineEdit, QFrame
from PyQt6.QtGui import QStandardItemModel, QStandardItem
from PyQt6.QtCore import Qt, QDate
from PyQt6.uic import loadUi
import sys
from datetime import datetime
from functools import partial
from db import get_people, insert_person, edit_person, delete_person, get_gifts, insert_gift, edit_gift,
delete_gift, get_people_with_reminders
from reminders import days_until_birthday, group_by_upcoming, notify_upcoming_birthdays


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        loadUi("ui/main.ui", self)
        self.setFixedSize(self.size())

        self.managePeopleButton.clicked.connect(self.open_manage_people_window)
        self.remindersButton.clicked.connect(self.open_reminders_window)
        self.giftsButton.clicked.connect(self.open_gift_ideas_window)
        self.exitButton.clicked.connect(self.quit_program)

    def open_manage_people_window(self):
        self.manage_people_window = ManagePeopleWindow(self)
        self.manage_people_window.show()
        self.hide()

    def open_reminders_window(self):
        self.reminders_window = RemindersWindow(self)
        self.reminders_window.show()
        self.hide()

    def open_gift_ideas_window(self):
        self.gift_ideas_window = GiftIdeasWindow(self)
        self.gift_ideas_window.show()
        self.hide()

    def quit_program(self):
        QApplication.quit()
        print("Program closed.")

class ManagePeopleWindow(QWidget):
    def __init__(self, main_window):
        super().__init__()
        loadUi("ui/manage_people.ui", self)
        self.setFixedSize(self.size())
        self.main_window = main_window

        self.load_people()
        self.addPersonButton.clicked.connect(self.open_add_person_window)
        self.editPersonButton.clicked.connect(self.open_edit_person_window)
        self.deletePersonButton.clicked.connect(self.delete_person)
        self.searchLineEdit.textChanged.connect(self.filter_people)
        self.backButton.clicked.connect(self.go_back)

    def load_people(self):
        self.people_data = get_people()

        if self.people_data is None:
            QMessageBox.critical(self, "Error", "Failed to load people from the database.")
            return
```

```
        model = QStandardItemModel()
        model.setHorizontalHeaderLabels(["ID", "First Name", "Last Name", "Birth Date", "Email", "Phone", "Notes"])

        for row in self.people_data:
            items = [QStandardItem(str(field)) for field in row]
            model.appendRow(items)

        self.peopleTableView.setModel(model)
        self.peopleTableView.setColumnHidden(0, True)
        self.peopleTableView.resizeColumnsToContents()

    def open_add_person_window(self):
        self.add_person_window = AddPersonWindow(callback=self.load_people)
        self.add_person_window.setWindowModality(Qt.WindowModality.ApplicationModal)
        self.add_person_window.show()

    def open_edit_person_window(self):
        selected_person = self.peopleTableView.selectedIndexes()
        if not selected_person:
            QMessageBox.warning(self, "Selection Error", "Please select a person to edit.")
            return

        person_id = self.peopleTableView.model().item(selected_person[0].row(), 0).text()
        first_name = self.peopleTableView.model().item(selected_person[0].row(), 1).text()
        last_name = self.peopleTableView.model().item(selected_person[0].row(), 2).text()
        birth_date = self.peopleTableView.model().item(selected_person[0].row(), 3).text()
        email = self.peopleTableView.model().item(selected_person[0].row(), 4).text()
        phone = self.peopleTableView.model().item(selected_person[0].row(), 5).text()
        notes = self.peopleTableView.model().item(selected_person[0].row(), 6).text()

         self.edit_person_window = EditPersonWindow(person_id, first_name, last_name, birth_date, email, phone,
notes, callback=self.load_people)
        self.edit_person_window.setWindowModality(Qt.WindowModality.ApplicationModal)
        self.edit_person_window.show()

    def delete_person(self):
        selected_person = self.peopleTableView.selectedIndexes()
        if not selected_person:
            QMessageBox.warning(self, "Selection Error", "Please select a person to delete.")
            return

        row = selected_person[0].row()
        person_id = self.peopleTableView.model().item(row, 0).text()
        first_name = self.peopleTableView.model().item(row, 1).text()
        last_name = self.peopleTableView.model().item(row, 2).text()

        confirm = QMessageBox.question(
            self,
            "Confirm Delete",
            f"Are you sure you want to delete {first_name} {last_name}?",
            QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No
        )

        if confirm == QMessageBox.StandardButton.Yes:
            if delete_person(person_id):
                QMessageBox.information(self, "Deleted", "Person deleted successfully.")
                print("Person deleted successfully.")
                self.load_people()
            else:
                QMessageBox.warning(self, "Error", "Failed to delete person.")
                print("Failed to delete person.")

    def filter_people(self, text):
        filtered = []
        text = text.lower()

        for row in self.people_data:
            if any(text in str(field).lower() for field in row[1:]):
                filtered.append(row)

        self.update_people_table(filtered)

    def update_people_table(self, data):
        model = QStandardItemModel()
        model.setHorizontalHeaderLabels(["ID", "First Name", "Last Name", "Birth Date", "Email", "Phone", "Notes"])

        for row in data:
            items = [QStandardItem(str(field)) for field in row]
            model.appendRow(items)

        self.peopleTableView.setModel(model)
        self.peopleTableView.setColumnHidden(0, True)
        self.peopleTableView.resizeColumnsToContents()

    def go_back(self):
        self.main_window.show()
        self.close()

class AddPersonWindow(QWidget):
    def __init__(self, callback=None):
        super().__init__()
        loadUi("ui/add_person.ui", self)
        self.setFixedSize(self.size())

        self.callback = callback

        self.addPersonSaveButton.clicked.connect(self.save_person)
```

```
        self.addPersonCancelButton.clicked.connect(self.cancel)

    def save_person(self):
        first_name = self.addFirstName.text()
        last_name = self.addLastName.text()
        email = self.addEmail.text()
        phone = self.addNumber.text()
        birth_date = self.addBirthDate.date().toString("yyyy-MM-dd")
        notes = self.addPersonNotes.toPlainText()

        if not first_name or not last_name:
            QMessageBox.warning(self, "Input Error", "First name and last name cannot be empty.")
            return

        print("Saving person:", first_name, last_name, birth_date, email, phone, notes)

        success = insert_person(first_name, last_name, birth_date, email, phone, notes)

        if success:
            QMessageBox.information(self, "Deleted", "Person added successfully.")
            print("Person added successfully.")
            if self.callback:
                self.callback()
            self.close()
        else:
            QMessageBox.warning(self, "Error", "Failed to add person.")
            print("Failed to add person.")

    def cancel(self):
        self.close()


class EditPersonWindow(QWidget):
    def __init__(self, person_id, first_name, last_name, birth_date, email, phone, notes, callback=None):
        super().__init__()
        loadUi("ui/edit_person.ui", self)
        self.setFixedSize(self.size())

        self.person_id = person_id
        self.callback = callback

        self.editFirstName.setText(first_name)
        self.editLastName.setText(last_name)
        self.editBirthDate.setDate(QDate.fromString(birth_date, "yyyy-MM-dd"))
        self.editEmail.setText(email)
        self.editNumber.setText(phone)
        self.editPersonNotes.setPlainText(notes)

        self.editPersonSaveButton.clicked.connect(self.save_person)
        self.editPersonCancelButton.clicked.connect(self.cancel)

    def save_person(self):
        first_name = self.editFirstName.text()
        last_name = self.editLastName.text()
        email = self.editEmail.text()
        phone = self.editNumber.text()
        birth_date = self.editBirthDate.date().toString("yyyy-MM-dd")
        notes = self.editPersonNotes.toPlainText()

        success = edit_person(first_name, last_name, birth_date, email, phone, notes, self.person_id)

        if success:
            QMessageBox.information(self, "Success", "Person information updated successfully.")
            print("Information updated successfully!")
            if self.callback:
                self.callback()
            self.close()
        else:
            QMessageBox.warning(self, "Error", "Failed to update information.")
            print("Failed to update information.")

    def cancel(self):
        self.close()

class GiftIdeasWindow(QWidget):
    def __init__(self, main_window):
        super().__init__()
        loadUi("ui/gift_ideas.ui", self)
        self.setFixedSize(self.size())
        self.main_window = main_window

        self.load_gifts()
        self.addGiftButton.clicked.connect(self.open_add_gift_window)
        self.editGiftButton.clicked.connect(self.open_edit_gift_window)
        self.deleteGiftButton.clicked.connect(self.delete_gift)
        self.searchLineEdit.textChanged.connect(self.filter_gifts)
        self.backButton.clicked.connect(self.go_back)

    def load_gifts(self):
        self.gifts_data = get_gifts()

        if self.gifts_data is False:
            QMessageBox.warning(self, "Load Error", "Could not load gifts data.")
            self.close()
            return

        model = QStandardItemModel()
```

```
        model.setHorizontalHeaderLabels(["ID", "Person", "Gift Name", "Description", "Place", "Link", "Price",
"Notes"])

        for row in self.gifts_data:
            items = [QStandardItem(str(field)) for field in row]
            model.appendRow(items)

        self.giftsTableView.setModel(model)
        self.giftsTableView.setColumnHidden(0, True)
        self.giftsTableView.resizeColumnsToContents()

    def open_add_gift_window(self):
        people_data = get_people()

        if people_data is False:
            QMessageBox.warning(self, "Load Error", "Could not load people data.")
            return

        if not people_data:
            QMessageBox.warning(self, "No People", "You have no people in the database.")
            return

        self.add_gifts_window = AddGiftsWindow(people_data=people_data, callback=self.load_gifts)
        self.add_gifts_window.setWindowModality(Qt.WindowModality.ApplicationModal)
        self.add_gifts_window.show()

    def open_edit_gift_window(self):

        if not self.gifts_data:
            QMessageBox.warning(self, "No Gifts", "You have no gifts in the database.")
            return

        selected_gift = self.giftsTableView.selectedIndexes()
        if not selected_gift:
            QMessageBox.warning(self, "Selection Error", "Please select a gift to edit.")
            return

        gift_id = self.giftsTableView.model().item(selected_gift[0].row(), 0).text()
        person = self.giftsTableView.model().item(selected_gift[0].row(), 1).text()
        gift_name = self.giftsTableView.model().item(selected_gift[0].row(), 2).text()
        description = self.giftsTableView.model().item(selected_gift[0].row(), 3).text()
        place = self.giftsTableView.model().item(selected_gift[0].row(), 4).text()
        link = self.giftsTableView.model().item(selected_gift[0].row(), 5).text()
        price = self.giftsTableView.model().item(selected_gift[0].row(), 6).text()
        notes = self.giftsTableView.model().item(selected_gift[0].row(), 7).text()

        self.edit_gifts_window = EditGiftsWindow(gift_id, person, gift_name, description, place, link, price,
notes, callback=self.load_gifts)
        self.edit_gifts_window.setWindowModality(Qt.WindowModality.ApplicationModal)
        self.edit_gifts_window.show()

    def delete_gift(self):

        if not self.gifts_data:
            QMessageBox.warning(self, "No Gifts", "You have no gifts in the database.")
            return

        selected_gift = self.giftsTableView.selectedIndexes()
        if not selected_gift:
            QMessageBox.warning(self, "Selection Error", "Please select a gift to delete.")
            return

        row = selected_gift[0].row()
        gift_id = self.giftsTableView.model().item(row, 0).text()
        person = self.giftsTableView.model().item(row, 1).text()
        gift_name = self.giftsTableView.model().item(row, 2).text()

        confirm = QMessageBox.question(
            self,
            "Confirm Delete",
            f"Are you sure you want to delete the gift '{gift_name}' for {person}?",
            QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No
        )

        if confirm == QMessageBox.StandardButton.Yes:
            if delete_gift(gift_id):
                QMessageBox.information(self, "Deleted", "Gift deleted successfully.")
                print("Gift deleted successfully.")
                self.load_gifts()
            else:
                QMessageBox.warning(self, "Error", "Failed to delete gift.")
                print("Failed to delete gift.")

    def filter_gifts(self, text):
        filtered = []
        text = text.lower()

        for row in self.gifts_data:
            if any(text in str(field).lower() for field in row[1:]):
                filtered.append(row)

        self.update_gifts_table(filtered)

    def update_gifts_table(self, data):
        model = QStandardItemModel()
        model.setHorizontalHeaderLabels(["ID", "Person", "Gift Name", "Description", "Place", "Link", "Price",
"Notes"])
```

```
        for row in data:
            items = [QStandardItem(str(field)) for field in row]
            model.appendRow(items)

        self.giftsTableView.setModel(model)
        self.giftsTableView.setColumnHidden(0, True)
        self.giftsTableView.resizeColumnsToContents()

    def go_back(self):
        self.main_window.show()
        self.close()

class AddGiftsWindow(QWidget):
    def __init__(self, people_data, callback=None):
        super().__init__()
        loadUi("ui/add_gift.ui", self)
        self.setFixedSize(self.size())

        self.people_data = people_data
        self.callback = callback

        self.populate_people_combobox()
        self.addGiftSaveButton.clicked.connect(self.save_gift)
        self.addGiftCancelButton.clicked.connect(self.cancel)

    def populate_people_combobox(self):
        self.addPersonComboBox.clear()
        for person in self.people_data:
            person_id, first_name, last_name, *_ = person
            full_name = f"{first_name} {last_name}"
            self.addPersonComboBox.addItem(full_name, userData=person_id)

    def save_gift(self):
        person_id = self.addPersonComboBox.currentData()
        gift_name = self.addGiftName.text().strip()
        description = self.addDescription.toPlainText().strip()
        place = self.addPlace.text().strip()
        link = self.addLink.text().strip()
        price = self.addPrice.text().strip()
        notes = self.addGiftNotes.toPlainText().strip()

        if not person_id or not gift_name:
            QMessageBox.warning(self, "Input Error", "Person and gift name cannot be empty.")
            return

        if price != "":
            try:
                price = float(price)
                if price < 0:
                    raise ValueError("Price cannot be negative.")
                if price > 1_000_000:
                    raise ValueError("Price is unrealistically high.")
            except ValueError as ve:
                QMessageBox.warning(self, "Invalid Price", f"Please enter a valid price.\n\n{ve}")
                return

        success = insert_gift(person_id, gift_name, description, place, link, price, notes)

        if success:
            QMessageBox.information(self, "Success", "Gift added successfully.")
            print("Gift added successfully.")
            if self.callback:
                self.callback()
            self.close()
        else:
            QMessageBox.warning(self, "Error", "Failed to add gift.")
            print("Failed to add gift.")

    def cancel(self):
        self.close()

class EditGiftsWindow(QWidget):
    def __init__(self, gift_id, person, gift_name, description, place, link, price, notes, callback=None):
        super().__init__()
        loadUi("ui/edit_gift.ui", self)
        self.setFixedSize(self.size())

        self.gift_id = gift_id
        self.callback = callback

        self.populate_people_combobox(person)

        self.editGiftName.setText(gift_name)
        self.editDescription.setPlainText(description)
        self.editPlace.setText(place)
        self.editLink.setText(link)
        self.editPrice.setText(price)
        self.editGiftNotes.setPlainText(notes)

        self.editGiftSaveButton.clicked.connect(self.save_gift)
        self.editGiftCancelButton.clicked.connect(self.cancel)

    def populate_people_combobox(self, current_person_name):
        self.people_data = get_people()
        if self.people_data is None:
            QMessageBox.warning(self, "No People", "You have no people in the database.")
```

```
            return

        self.editPersonComboBox.clear()

        selected_index = 0
        for i, person in enumerate(self.people_data):
            person_id, first_name, last_name, *_ = person
            full_name = f"{first_name} {last_name}"
            self.editPersonComboBox.addItem(full_name, userData=person_id)

            if full_name == current_person_name:
                selected_index = i

        self.editPersonComboBox.setCurrentIndex(selected_index)

    def save_gift(self):
        person_id = self.editPersonComboBox.currentData()
        gift_name = self.editGiftName.text().strip()
        description = self.editDescription.toPlainText().strip()
        place = self.editPlace.text().strip()
        link = self.editLink.text().strip()
        price = self.editPrice.text().strip()
        notes = self.editGiftNotes.toPlainText().strip()

        if not person_id or not gift_name:
            QMessageBox.warning(self, "Input Error", "Person and gift name cannot be empty.")
            return

        try:
            price = float(price)
            if price < 0:
                raise ValueError("Price cannot be negative.")
            if price > 1_000_000:
                raise ValueError("Price is unrealistically high.")
        except ValueError as ve:
            QMessageBox.warning(self, "Invalid Price", f"Please enter a valid price.\n\n{ve}")
            return

        success = edit_gift(person_id, gift_name, description, place, link, price, notes, self.gift_id)

        if success:
            QMessageBox.information(self, "Success", "Gift updated successfully.")
            print("Gift updated successfully.")
            if self.callback:
                self.callback()
            self.close()
        else:
            QMessageBox.warning(self, "Error", "Failed to update gift.")
            print("Failed to update gift.")

    def cancel(self):
        self.close()

class RemindersWindow(QWidget):
    def __init__(self, main_window):
        super().__init__()
        loadUi("ui/reminders.ui", self)
        self.setFixedSize(self.size())
        self.main_window = main_window

        self.backButton.clicked.connect(self.go_back)
        self.load_reminders()

    def load_reminders(self):
        people = get_people_with_reminders()
        if not people:
            QMessageBox.warning(self, "No Data", "No reminder-enabled people found.")
            return

        grouped = group_by_upcoming(people)

        self.populate_group(self.todayFrame, grouped["Today"])
        self.populate_group(self.thisWeekFrame, grouped["This Week"])
        self.populate_group(self.nextWeekFrame, grouped["Next Week"])
        self.populate_group(self.upcomingFrame, grouped["Upcoming"])

        notify_upcoming_birthdays(grouped)

    def create_vline(self):
        line = QFrame()
        line.setFrameShape(QFrame.Shape.VLine)
        line.setFrameShadow(QFrame.Shadow.Sunken)
        return line

    def populate_group(self, frame, group_data):
        layout = frame.layout()
        self.clear_layout(layout)

        for row, (person, _) in enumerate(group_data):
            name = f"{person['first_name']} {person['last_name']}"
            days_left = days_until_birthday(person['birth_date'])
            person_id = person['person_id']

            birth_date = datetime.strptime(person["birth_date"], "%Y-%m-%d")
            formatted_date = birth_date.strftime("%B %#d")

            name_label = QLabel(name)
```

```
            date_label = QLabel(formatted_date)
            days_left_label = QLabel(f"{days_left} days left")

            layout.addWidget(name_label, row, 0)
            layout.addWidget(self.create_vline(), row, 1)
            layout.addWidget(date_label, row, 2)
            layout.addWidget(self.create_vline(), row, 3)
            layout.addWidget(days_left_label, row, 4)

    def clear_layout(self, layout):
        while layout.count():
            item = layout.takeAt(0)
            widget = item.widget()
            if widget:
                widget.deleteLater()

    def go_back(self):
        self.main_window.show()
        self.close()

if __name__ == "__main__":
    try:
        app = QApplication(sys.argv)
        ui = MainWindow()
        ui.show()
        app.exec()
    except Exception as e:
        print("Fatal error:", e)
```

B. **db.py**

```
import sqlite3

def get_connection():
    conn = sqlite3.connect("db/bday_tracker.db")
    conn.execute("PRAGMA foreign_keys = ON")
    return conn

def get_people():
    try:
        con = get_connection()
        cursor = con.cursor()
         cursor.execute("SELECT person_id, first_name, last_name, birth_date, email, phone_number, notes FROM
people")
        people_data = cursor.fetchall()
        return people_data

    except Exception as e:
        print("Get people failed:", e)
        return False

    finally:
        con.close()

def insert_person(first_name, last_name, birth_date, email, phone, notes):
    try:
        conn = get_connection()
        cursor = conn.cursor()
        cursor.execute("""
            INSERT INTO people (first_name, last_name, birth_date, email, phone_number, notes)
            VALUES (?, ?, ?, ?, ?, ?)
        """, (first_name, last_name, birth_date, email, phone, notes))
        conn.commit()

        person_id = cursor.lastrowid

        cursor.execute("INSERT INTO reminders (person_id) VALUES (?)", (person_id,))

        conn.commit()
        return True

    except Exception as e:
        print("Insert failed:", e)
        return False

    finally:
        conn.close()

def edit_person(first_name, last_name, birth_date, email, phone, notes, person_id):
    try:
        con = get_connection()
        cursor = con.cursor()
        cursor.execute("""
            UPDATE people
            SET first_name = ?, last_name = ?, birth_date = ?, email = ?, phone_number = ?, notes = ?
            WHERE person_id = ?
        """, (first_name, last_name, birth_date, email, phone, notes, person_id))
        con.commit()
        return True

    except Exception as e:
        print("Edit failed:", e)
        return False

    finally:
```

```
        con.close()

def delete_person(person_id):
    try:
        con = get_connection()
        cursor = con.cursor()
        cursor.execute("DELETE FROM people WHERE person_id = ?", (person_id,))
        con.commit()
        return True

    except Exception as e:
        print("Deletion failed:", e)
        return False

    finally:
        con.close()

def get_gifts():
    try:
        con = get_connection()
        cursor = con.cursor()
        cursor.execute("""
            SELECT
                g.gift_id,
                p.first_name || ' ' || p.last_name AS full_name,
                g.gift_name,
                g.description,
                g.place,
                g.link,
                g.price,
                g.notes
            FROM gift_ideas g
            JOIN people p ON g.person_id = p.person_id
        """)
        gifts_data = cursor.fetchall()
        return gifts_data

    except Exception as e:
        print("Get gifts failed:", e)
        return False

    finally:
        con.close()

def insert_gift(person_id, gift_name, description, place, link, price, notes):
    try:
        conn = get_connection()
        cursor = conn.cursor()
        cursor.execute("""
            INSERT INTO gift_ideas (person_id, gift_name, description, place, link, price, notes)
            VALUES (?, ?, ?, ?, ?, ?, ?)
        """, (person_id, gift_name, description, place, link, price, notes))
        conn.commit()
        return True

    except Exception as e:
        print("Insert failed:", e)
        return False

    finally:
        conn.close()

def edit_gift(person_id, gift_name, description, place, link, price, notes, gift_id):
    try:
        conn = get_connection()
        cursor = conn.cursor()
        cursor.execute("""
            UPDATE gift_ideas
            SET person_id = ?, gift_name = ?, description = ?, place = ?, link = ?, price = ?, notes = ?
            WHERE gift_id = ?
        """, (person_id, gift_name, description, place, link, price, notes, gift_id))
        conn.commit()
        return True

    except Exception as e:
        print("Edit gift failed:", e)
        return False

    finally:
        conn.close()

def delete_gift(gift_id):
    try:
        con = get_connection()
        cursor = con.cursor()
        cursor.execute("DELETE FROM gift_ideas WHERE gift_id = ?", (gift_id,))
        con.commit()
        return True

    except Exception as e:
        print("Deletion failed:", e)
        return False

    finally:
        con.close()

def get_people_with_reminders():
```

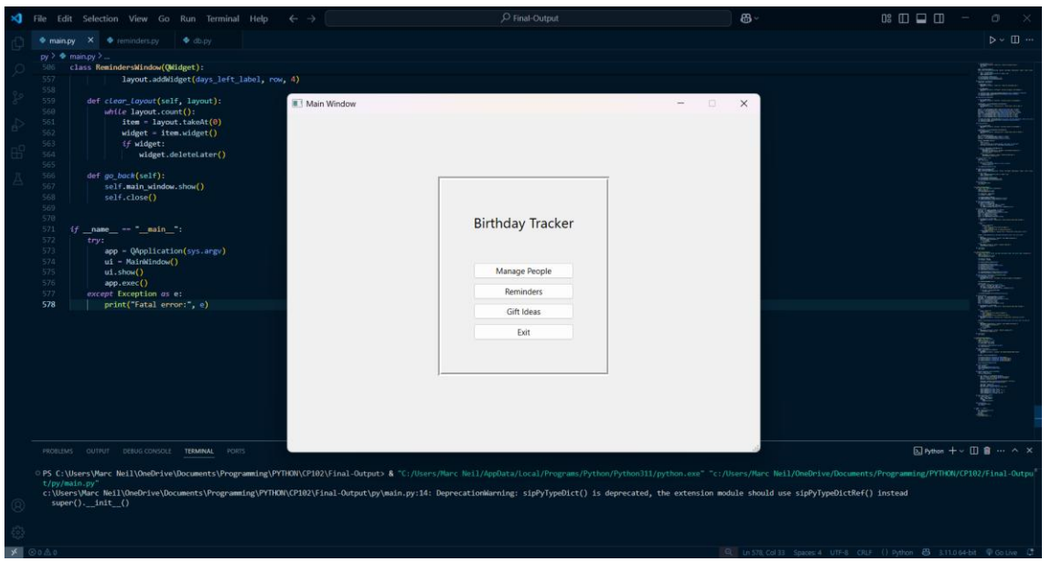
```
    try:
        con = get_connection()
        cursor = con.cursor()

        cursor.execute("""
            SELECT p.person_id, p.first_name, p.last_name, p.birth_date
            FROM people p
            JOIN reminders r ON p.person_id = r.person_id
        """)

        rows = cursor.fetchall()

        people = []
        for row in rows:
            person = {
                "person_id": row[0],
                "first_name": row[1],
                "last_name": row[2],
                "birth_date": row[3],
            }
            people.append(person)

        return people

    except Exception as e:
        print("Get reminders failed:", e)
        return False

    finally:
        con.close()
```

C. **reminders,py**

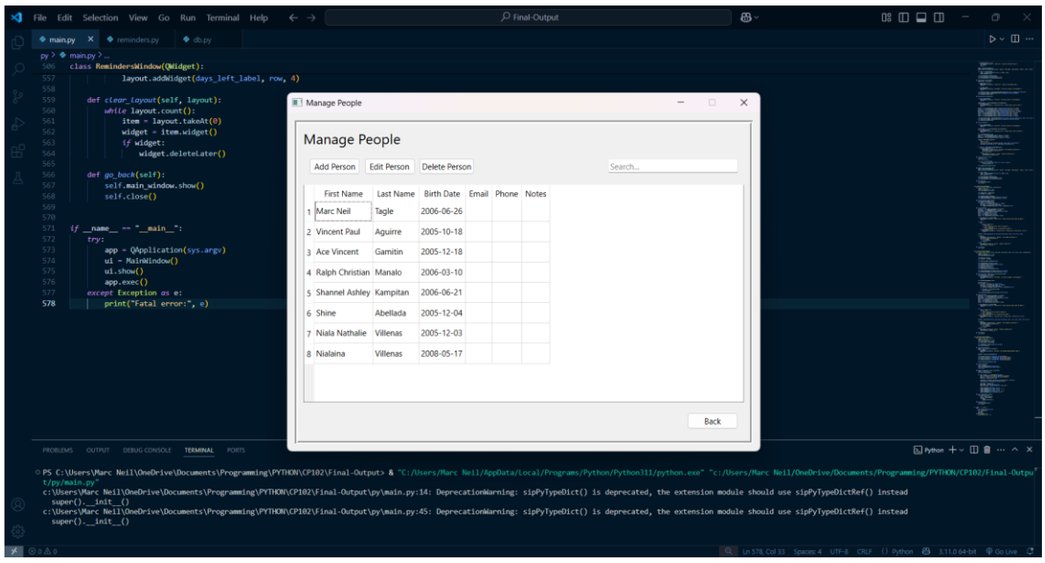
```
from datetime import datetime
from plyer import notification

def days_until_birthday(birth_date_str):
    today = datetime.today().date()
    birth_date = datetime.strptime(birth_date_str, "%Y-%m-%d").date()
    this_year_birthday = birth_date.replace(year=today.year)

    if this_year_birthday < today:
        next_birthday = this_year_birthday.replace(year=today.year + 1)
    else:
        next_birthday = this_year_birthday

    return (next_birthday - today).days

def group_by_upcoming(birthdays):
    grouped = {
        "Today": [],
        "This Week": [],
        "Next Week": [],
        "Upcoming": []
    }

    for person in birthdays:
        days_left = days_until_birthday(person['birth_date'])

        if days_left == 0:
            grouped["Today"].append((person, days_left))
        elif 1 <= days_left <= 7:
            grouped["This Week"].append((person, days_left))
        elif 8 <= days_left <= 14:
            grouped["Next Week"].append((person, days_left))
        else:
            grouped["Upcoming"].append((person, days_left))

    return grouped

def notify_upcoming_birthdays(grouped):
    for group in ["Today", "This Week", "Next Week"]:
        people_msgs = []
        for person, days_left in grouped.get(group, []):
            if not person.get("enabled", True):
                continue

            name = f"{person['first_name']} {person['last_name']}"

            if group == "Today":
                people_msgs.append(f"{name}")
            else:
                people_msgs.append(f"{name} ({days_left} {'day' if days_left == 1 else 'days'})")

            print("NOTIFICATION:", people_msgs)

        if people_msgs:
            message = "\n".join(people_msgs)
            notification.notify(
                title=f"Birthdays {group}",
                message=message,
                timeout=10,
                app_name="Birthday Tracker"
            )
```

Screenshots of Output

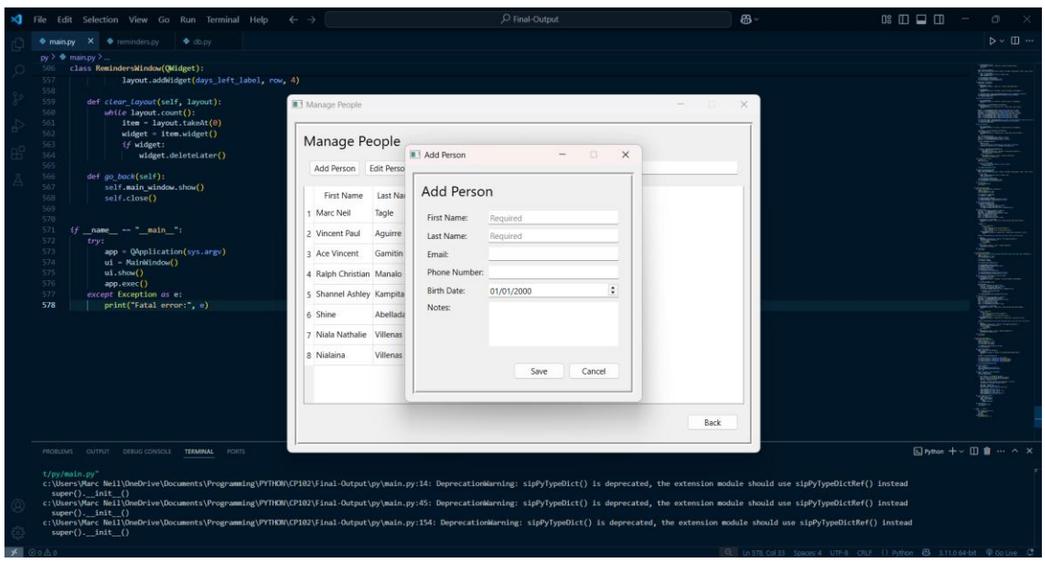
1. Main Menu



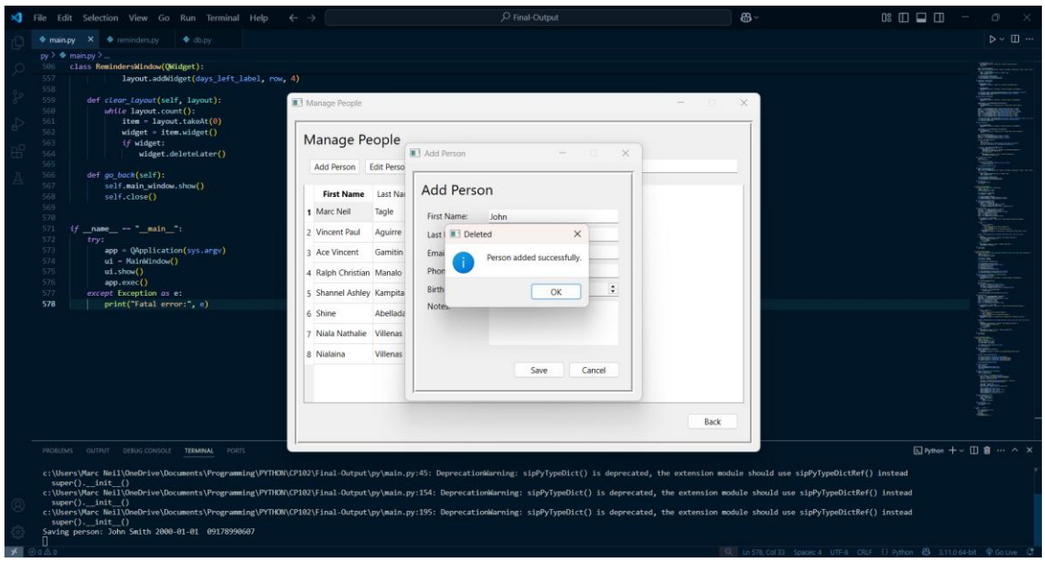
2. Manage People Window



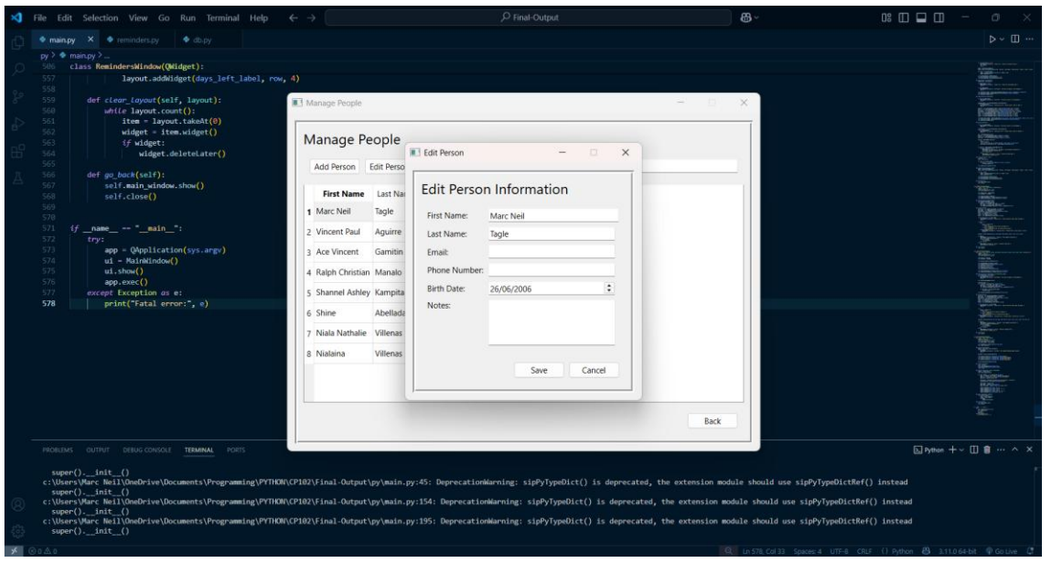
3. Add Person Window



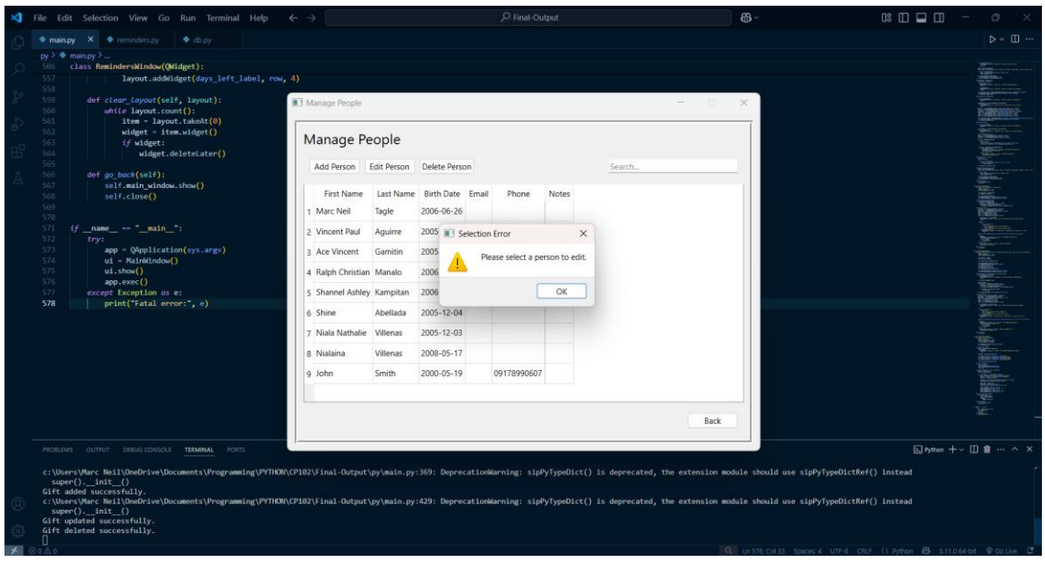
4. Successfully Adding a Person



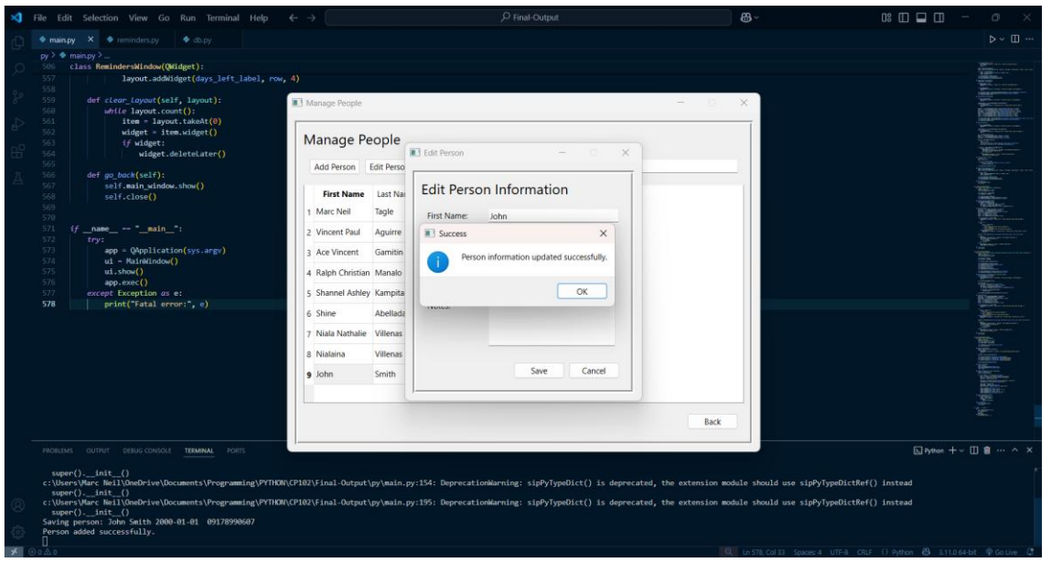
5. Edit Person Window



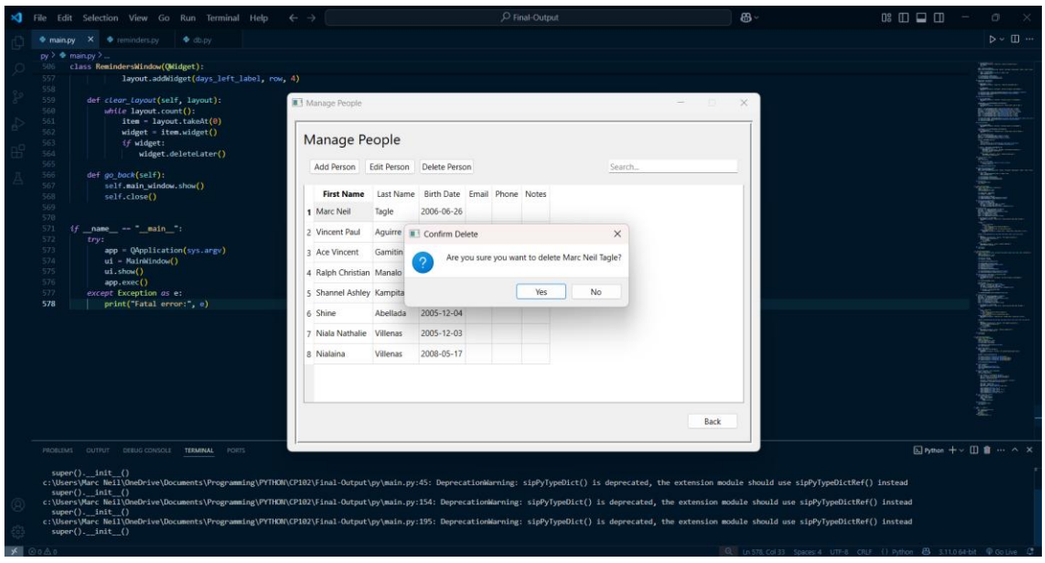
6. Person Selection Warning (Editing Person Information)



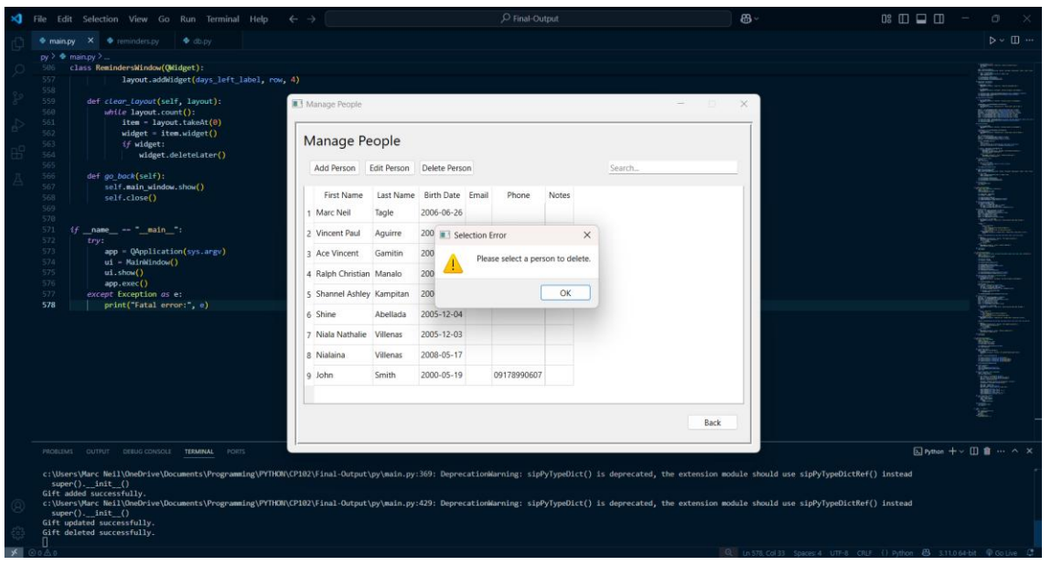
7. Successfully Updating a Person's Information



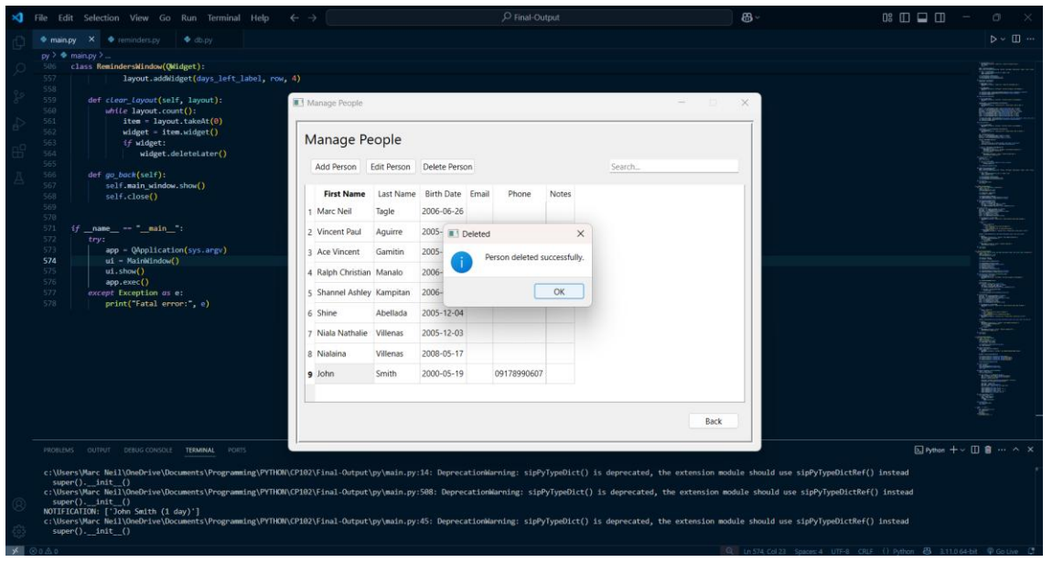
8. Delete Person Confirmation



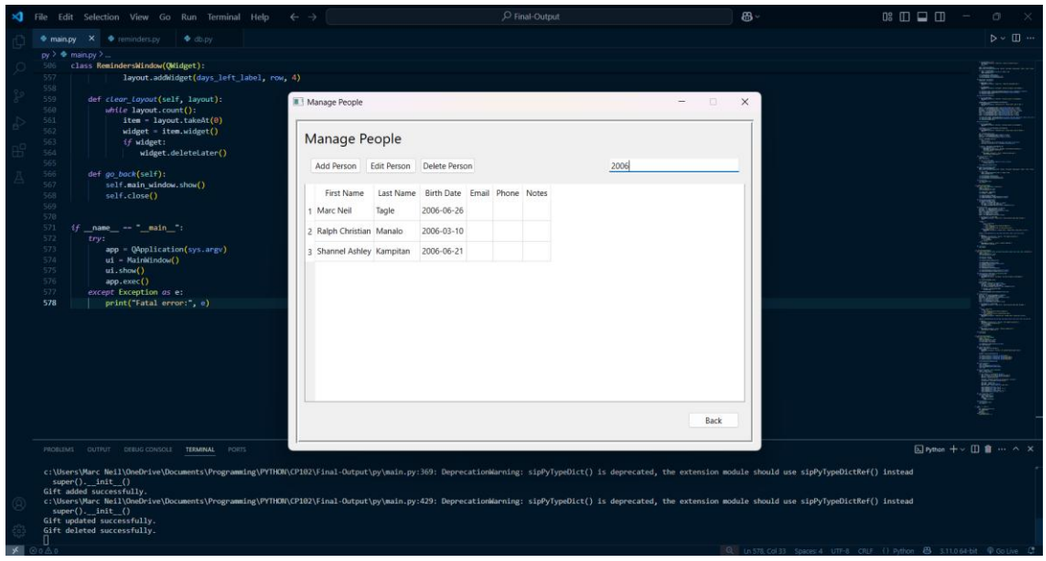
9. Person Selection Warning (Deleting Person)



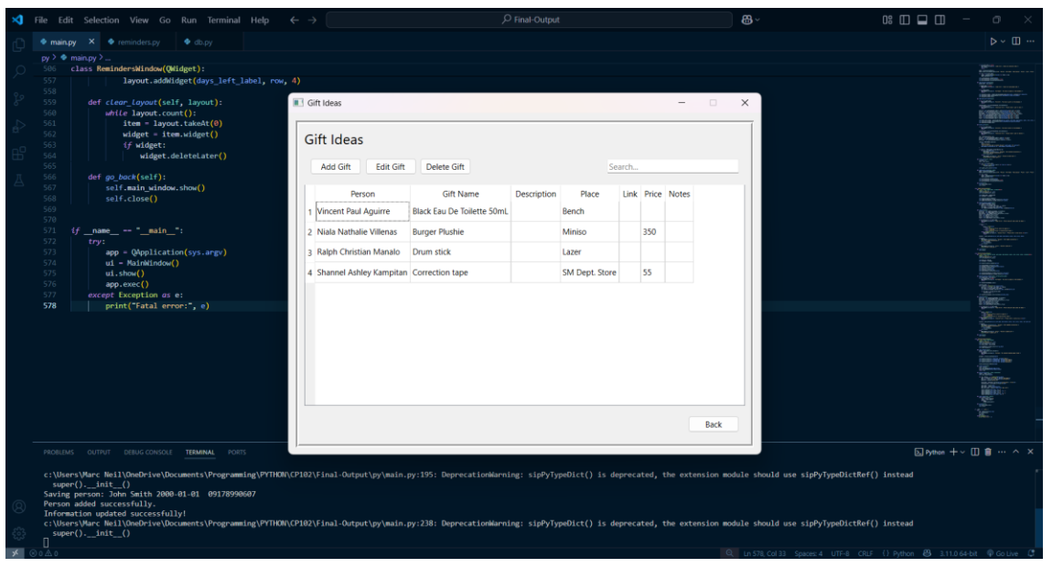
10. Successfully Deleting Person



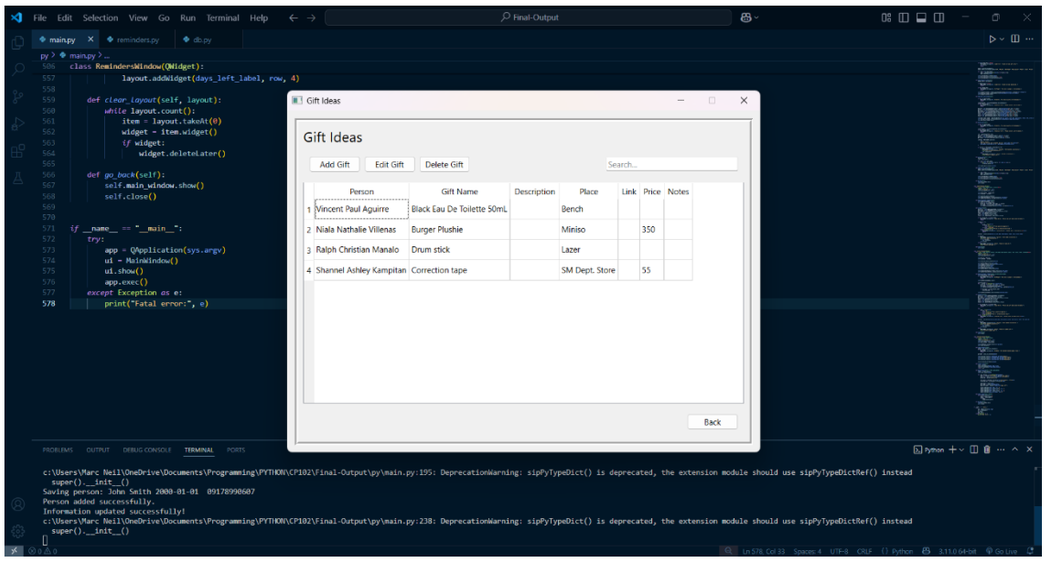
11. Using Person Search Bar (Can be used to search by any column/field)



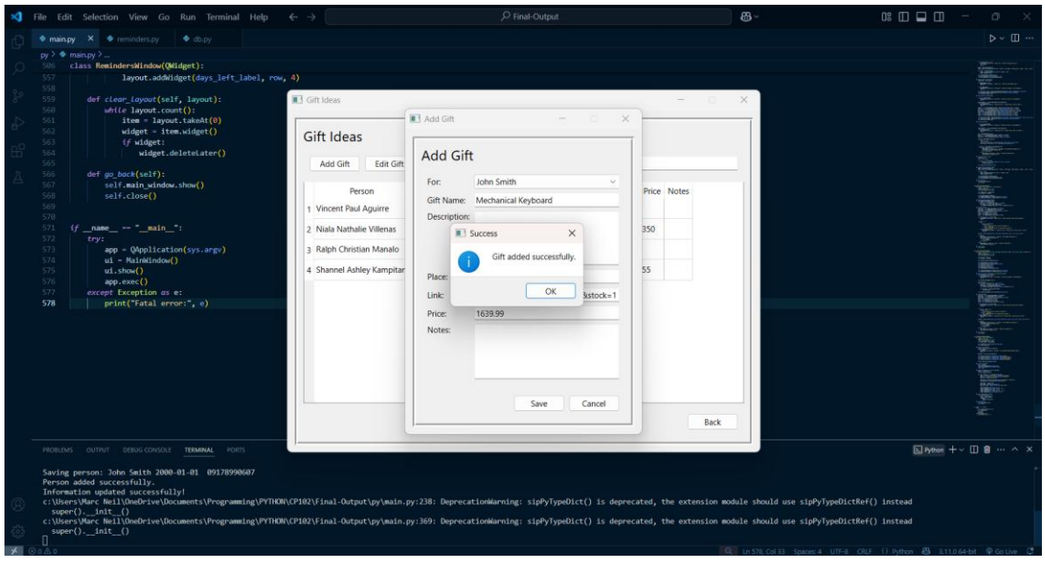
12. Gift Ideas Window



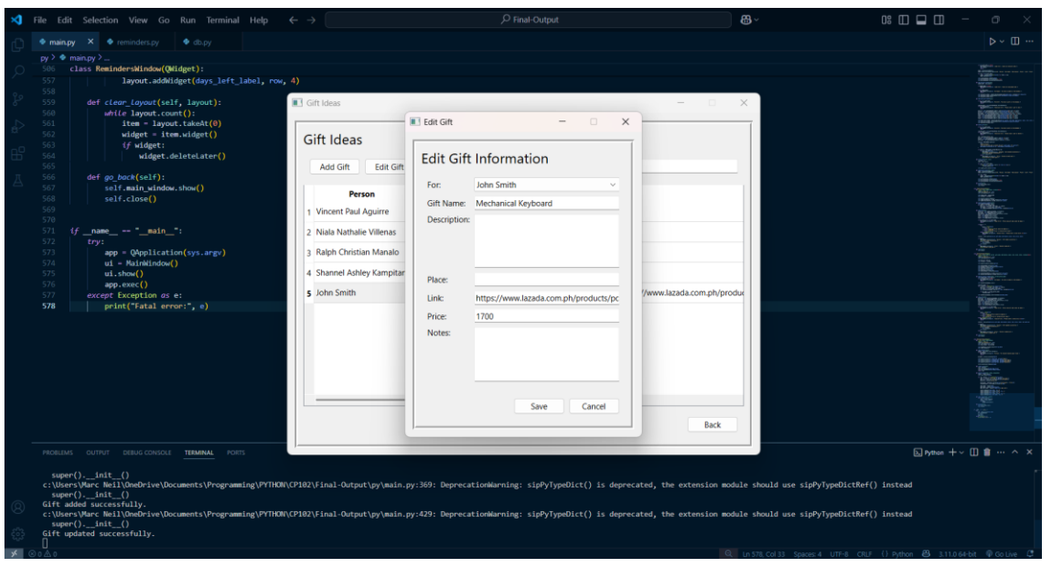
13. Add Gift Window



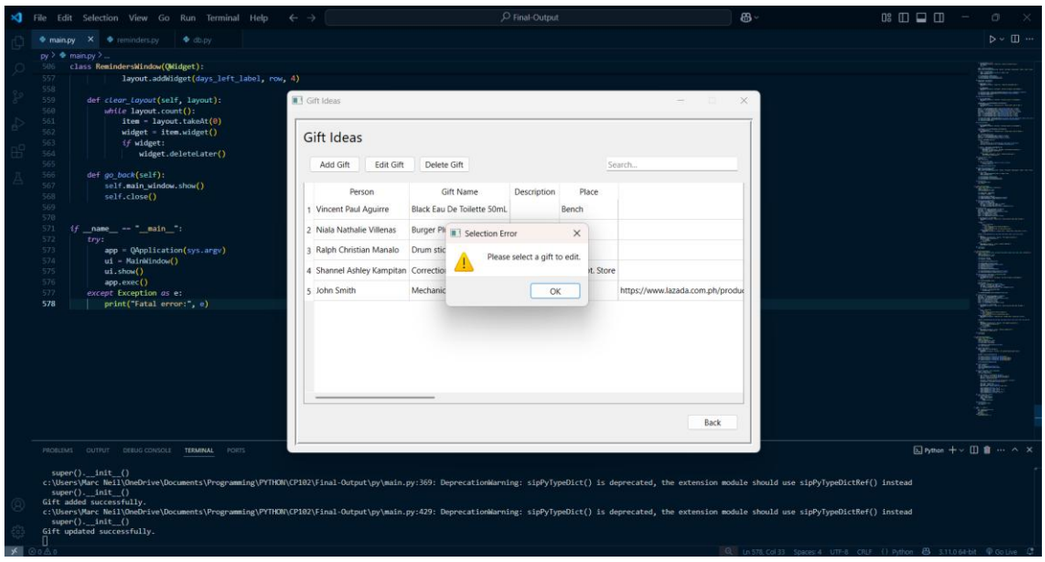
14. Successfully Adding a Gift Idea



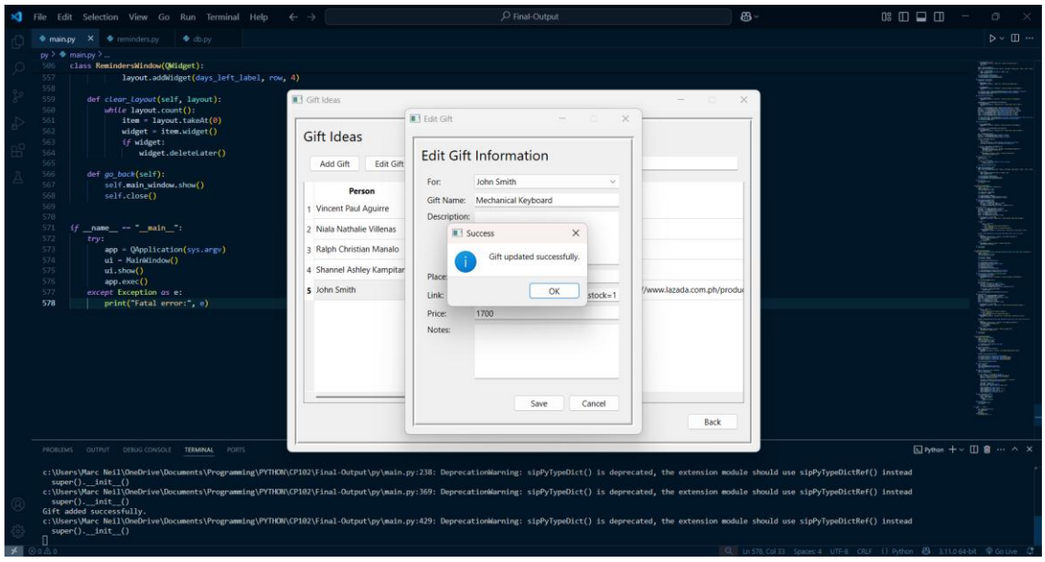
15. Edit Gift Window



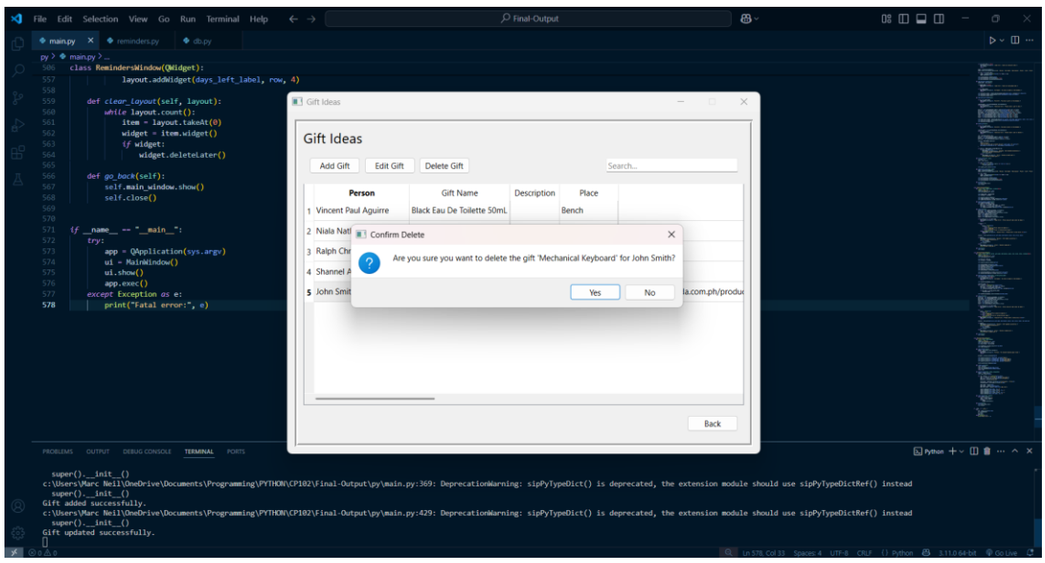
16. Gift Selection Warning (Editing Gift Information)



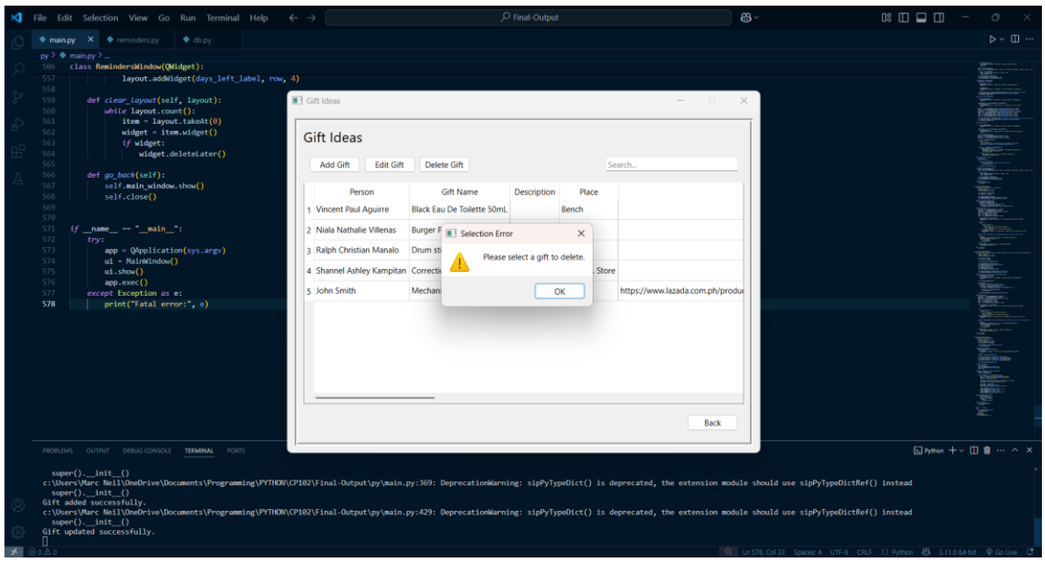
17. Successfully Updating Gift Idea Information



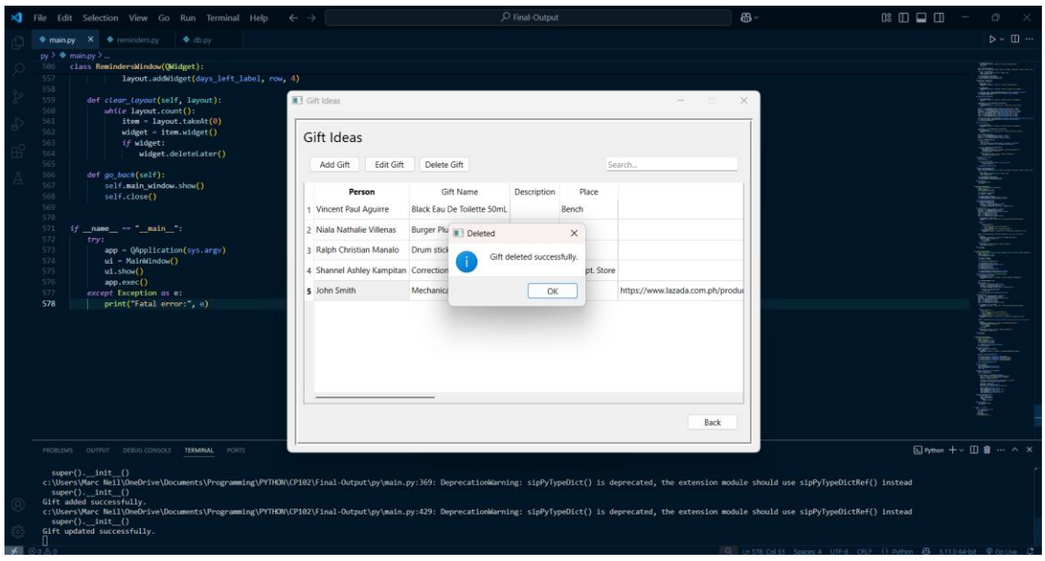
18. Delete Gift Confirmation



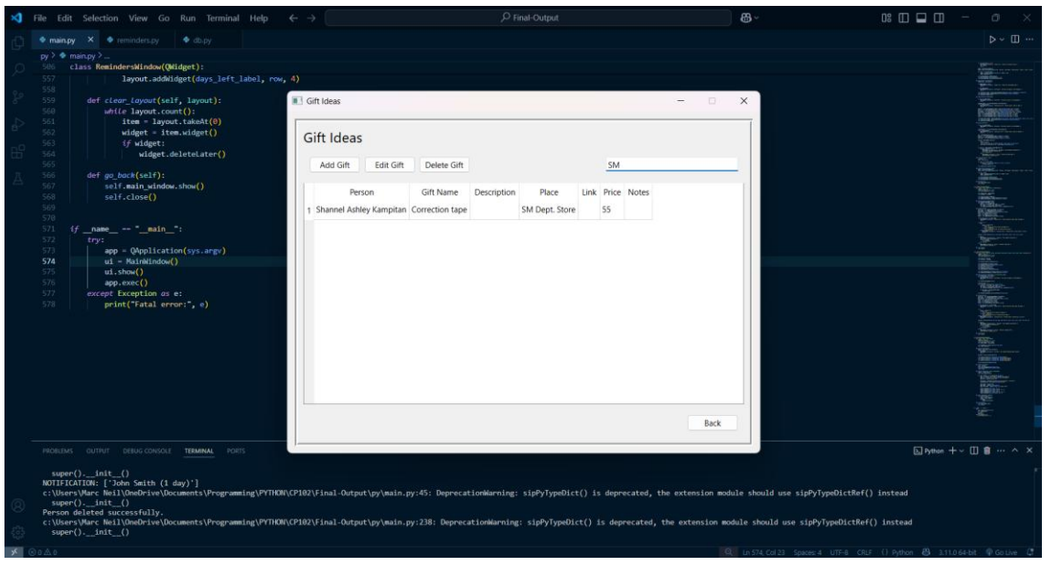
19. Gift Selection Warning (Deleting Gift)



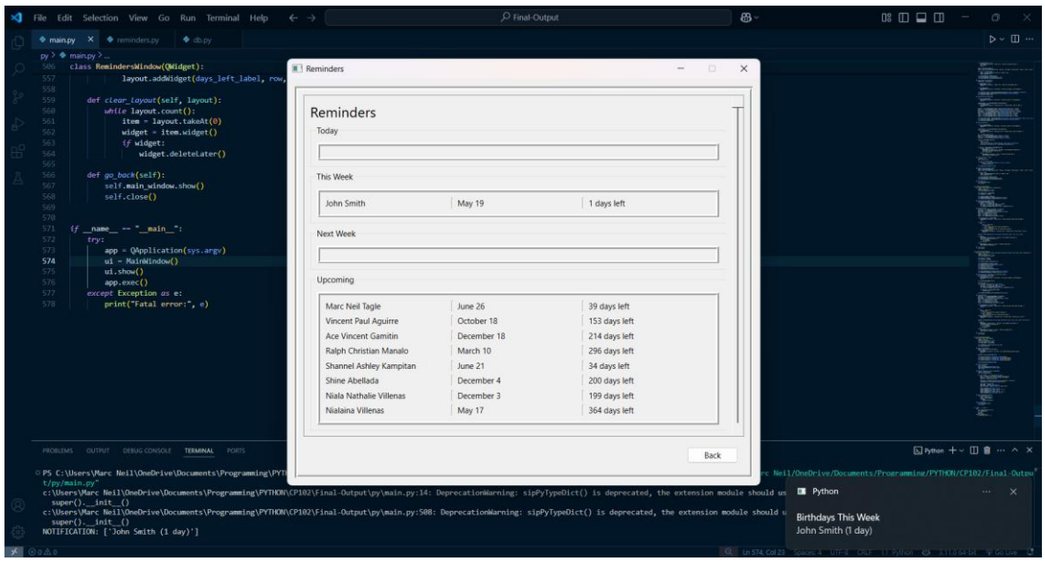
20. Successfully Deleting a Gift Idea



21. Using Gift Ideas Search Bar (Can be used to search by any column/field)



22. Reminders Window (Windows notification upon opening window)



Learning Reflection

CP102 has been a *meaningful chapter* in my programming journey. As a continuation of CP101, this course introduced us to *more advanced concepts in Python* such as file handling, data analysis with Pandas, data visualization using hvPlot, working with CSV files, GUI development with Tkinter and PyQt6, object-oriented programming (OOP), and MySQL database integration. These topics provided a deeper and more practical understanding of how programming is used to solve *real-world problems*.

To be honest, there were moments when I *struggled* to fully grasp some of the concepts, especially when the code became more complex. In those times, I turned to *AI tools* to help me understand better, whether by breaking down lines of code or explaining unfamiliar functions. While these tools were helpful, I realized that programming still demands *initiative, personal effort, and dedication*. No amount of assistance replaces the need to put in the work, practice consistently, and debug through trial and error.

There were times of *frustration*, but what matters is that I *continue to learn and improve*. I did not expect to come this far already, and being able to accomplish this course feels like a true achievement. I look up to my seniors who have already accomplished so much, and I hope that someday, I would be successful too.

I won't lie. I'm scared of what the future holds. The road ahead in computer science looks challenging, but I am also *excited and motivated* to keep enhancing my skills. I want to build more meaningful and fulfilling projects that I can be proud of.

Lastly, I would like to thank my professor, **Dean Belleza**, for this opportunity and for generously devoting his time to share his knowledge with us. His guidance throughout the course made a *significant impact* on my learning experience.