# Date & Time Applications - Project Report

## Problem Statement

Modern applications often require date and time functionality for various purposes such as calculating ages, determining the day of the week for a given date, and counting down to important events. Users need a unified, easy-to-use tool that consolidates these three commonly needed operations into a single menu-driven interface.

The challenge was to develop a Python program that:

- Accepts user input in a consistent date format (YYYY-MM-DD)
- Performs accurate date calculations using Python's datetime library
- Handles invalid inputs gracefully without crashing
- Provides an intuitive menu-based interface for selecting between three different date/time operations
- Delivers precise results for age calculation, day-of-week lookup, and event countdown
- Implements a graphical user interface (GUI) for the countdown feature using wxPython

This project demonstrates core programming concepts including input validation, error handling, logical computation, and object-oriented principles through the use of Python's datetime module and wxPython framework for GUI development.

---

# Approach Used

## Architecture Overview

The solution follows a **menu-driven, modular approach** where each date/time operation is implemented as a separate function, called from a central menu loop. This architecture ensures code reusability, maintainability, and clear separation of concerns. The countdown feature is enhanced with a graphical interface using wxPython for improved user experience.

## Module: datetime Library

The Python `datetime` module provides built-in classes for handling dates and times:

- **datetime class**: Represents a specific date and time
- **date class**: Represents just the date (year, month, day)
- **strptime()**: Parses a date string into a datetime object using a format pattern
- **strftime()**: Formats a datetime object back into a string

## Module: wxPython GUI Framework

wxPython is a Python binding for the wxWidgets C++ library, providing:

- **wx.Frame**: Main window container for the application
- **wx.Panel**: Container for GUI elements
- **wx.TextCtrl**: Text input fields for user data
- **wx.Button**: Clickable buttons for actions
- **wx.StaticText**: Labels and display text
- **wx.BoxSizer**: Layout management for responsive design
- Event binding for interactive user interactions

# Implementation Strategy

## 1. Day of Week Calculator

**Algorithm:**

1. Prompt user for a date string in format YYYY-MM-DD
2. Use `datetime.strptime()` to convert string to datetime object
3. Use `strftime("%A")` to extract the full weekday name
4. Display the result to the user

**Key Code:**

```
dt = datetime.strptime(date_str, "%Y-%m-%d")
print("Day of the week:", dt.strftime("%A"))
```

**Why this works:** Python's datetime library internally manages calendar rules (leap years, month lengths) so the calculation is always accurate.

---

## 2. Age Calculator

**Algorithm:**

1. Prompt user for their birth date in YYYY-MM-DD format
2. Parse birth date and get today's date using `date.today()`
3. Calculate initial age: `today.year - birth.year`
4. Check if birthday has occurred this year using tuple comparison
5. If birthday hasn't occurred yet, subtract 1 from the calculated age

**Key Code:**

```
years = today.year - birth.year - ((today.month, today.day) < (birth.month, birth.day))
```

**Why this approach:** Simply subtracting years would give inaccurate results for people who haven't had their birthday yet in the current year. The tuple comparison `(today.month, today.day) < (birth.month, birth.day)` returns `True` (value 1) if the birthday is pending, which is subtracted to adjust the age.

**Example:**

- Today: December 1, 2025
- Birth date: September 14, 2005
- Year difference: 2025 - 2005 = 20
- Has birthday passed? (12, 1) < (9, 14)? = False (0)
- Final age: 20 - 0 = **20 years old** ✓

---

## 3. Countdown to Future Event (with wxPython GUI)

**Algorithm:**

1. Launch a wxPython GUI window with input fields and buttons
2. Prompt user for an event date in YYYY-MM-DD format
3. Parse event date and get today's date
4. Calculate the difference: `event_date - today_date`
5. Extract the number of days from the difference
6. Display result in a graphical window with formatted output

**Key Code:**

```
import wx
from datetime import datetime, date

class CountdownFrame(wx.Frame):
    def __init__(self):
        super().__init__(None, title="Event Countdown Calculator", size=(400, 300))
        panel = wx.Panel(self)
        sizer = wx.BoxSizer(wx.VERTICAL)

        # Input label and field
        label = wx.StaticText(panel, label="Enter event date (YYYY-MM-DD):")
        self.input_field = wx.TextCtrl(panel, size=(300, 25))

        # Calculate button
        calculate_btn = wx.Button(panel, label="Calculate")
        calculate_btn.Bind(wx.EVT_BUTTON, self.on_calculate)

        # Result display
        self.result = wx.StaticText(panel, label="")

        # Add to sizer
        sizer.Add(label, 0, wx.ALL, 10)
        sizer.Add(self.input_field, 0, wx.ALL, 10)
        sizer.Add(calculate_btn, 0, wx.ALL, 10)
        sizer.Add(self.result, 0, wx.ALL, 10)

        panel.SetSizer(sizer)

    def on_calculate(self, event):
        try:
            event_date = datetime.strptime(self.input_field.GetValue(), "%Y-%m-%d").date()
            today = date.today()
            delta = (event_date - today).days

            if delta > 0:
                self.result.SetLabel(f"Days left: {delta}")
            elif delta == 0:
                self.result.SetLabel("The event is today!")
            else:
                self.result.SetLabel(f"Event was {-delta} days ago")
        except ValueError:
            self.result.SetLabel("Invalid format! Use YYYY-MM-DD")
```

**Why wxPython:** Provides a modern, interactive GUI that improves user experience by offering:

- Real-time feedback without console-based input/output
- Cleaner visual presentation
- Better accessibility for non-technical users
- Cross-platform compatibility (Windows, Mac, Linux)

---

# Menu-Driven Loop (Console Interface)

The main program uses a `while True` loop that continuously displays the menu until the user selects "Exit":

```
def main():
    while True:
        print("\nDate & Time Menu")
        print("1. Day of the week calculator")
        print("2. Age calculator")
        print("3. Countdown to a future event (GUI)")
        print("4. Exit")
        choice = input("Enter your choice (1-4): ")

        if choice == "1":
            day_of_week_calculator()
        elif choice == "2":
            age_calculator()
        elif choice == "3":
            app = wx.App()
            frame = CountdownFrame()
            frame.Show()
            app.MainLoop()
        elif choice == "4":
            break
        else:
            print("Invalid choice. Please enter 1-4.")
```

# Error Handling Strategy

Each function uses a **try-except block** to handle invalid date inputs:

- **Try block**: Attempts to parse user input using `strptime()`
- **Except ValueError**: Catches format errors and displays helpful message
- User is then returned to the menu to try again

---

# Sample Input and Output

# Example 1: Day of Week Calculator

**User Input:**

```
Enter your choice (1-4): 1
Enter a date (YYYY-MM-DD): 2024-07-04
```

**Program Output:**

```
Day of the week: Thursday
```

**Verification:** July 4, 2024 was indeed a Thursday.

---

# Example 2: Age Calculator

**User Input:**

```
Enter your choice (1-4): 2
Enter your birth date (YYYY-MM-DD): 2005-09-14
```

**Program Output:**

```
Your age is: 19 years
```

**Calculation Breakdown:**

- Today: December 1, 2025
- Birth: September 14, 2005
- Year difference: 2025 - 2005 = 20
- Birthday check: (12, 1) < (9, 14)? = False
- Final age: 20 - 0 = **19 years** (will turn 20 on September 14, 2026)

---

# Example 3: Countdown to Future Event (wxPython GUI)

**User Input:**

```
Enter your choice (1-4): 3
[wxPython window opens]
Enter event date (YYYY-MM-DD): 2025-12-25
[User clicks "Calculate" button]
```

**Program Output (GUI Display):**

```
Days left: 24
```

**Calculation:** December 25, 2025 - December 1, 2025 = 24 days

**GUI Features:**

- Clean, centered window with input field
- Real-time button response
- Formatted result display
- Cross-platform compatibility

---

# Example 4: Invalid Input Handling

**User Input:**

```
Enter your choice (1-4): 2
Enter your birth date (YYYY-MM-DD): 2005/09/14
```

**Program Output:**

```
Invalid date format. Use YYYY-MM-DD.
```

**Behavior:** Program returns to menu and allows user to retry.

---

# Example 5: Past Event Countdown (GUI)

**User Input (in wxPython window):**

```
Enter event date (YYYY-MM-DD): 2025-01-01
[Click Calculate]
```

**Program Output (GUI Display):**

```
Event was 334 days ago
```

**Calculation:** January 1, 2025 - December 1, 2025 = -334 days

---

# Challenges Faced and Solutions

## Challenge 1: Accurate Age Calculation

**Problem:** Simply subtracting birth year from current year gives incorrect results if the person hasn't had their birthday yet in the current year.

**Example:** If someone born September 14, 2005 is asked on December 1, 2025, simple subtraction (2025 - 2005 = 20) would show age 20, but they are still 19 until their next birthday.

**Solution:** Implement tuple comparison to detect if the birthday has passed:

```
years = today.year - birth.year - ((today.month, today.day) < (birth.month, birth.day))
```

The expression `((today.month, today.day) < (birth.month, birth.day))` returns:

- `True` (value 1) if birthday hasn't occurred yet → subtract 1
- `False` (value 0) if birthday has occurred → subtract 0

This elegant solution leverages Python's boolean-to-integer conversion.

---

# Challenge 2: Input Validation and Error Handling

**Problem:** Users might enter dates in incorrect formats (e.g., `2005/09/14` instead of `2005-09-14`), causing the program to crash with an unhandled exception.

**Solution:** Wrap date parsing in a try-except block:

```
try:
    birth = datetime.strptime(birth_str, "%Y-%m-%d").date()
except ValueError:
    print("Invalid date format. Use YYYY-MM-DD.")
```

This gracefully handles errors and prompts the user to retry with correct format. In wxPython GUI, errors are displayed in the result label without closing the window.

---

# Challenge 3: Leap Year and Month Length Complexity

**Problem:** Different months have different numbers of days (28, 29, 30, 31), and leap years affect February. Manual calculation would be error-prone.

**Solution:** Delegate all date arithmetic to Python's `datetime` module, which handles:

- Leap years automatically (every 4 years, except centuries unless divisible by 400)
- Variable month lengths
- Date validation (prevents invalid dates like February 30)

By using `datetime.strptime()` and date subtraction, all these complexities are handled transparently.

---

# Challenge 4: Menu Loop Control and User Experience

**Problem:** How to keep the program running until the user explicitly chooses to exit, while handling invalid menu choices. Additionally, integrating GUI with console-based menu required proper event loop management.

**Solution:** Implement a `while True` loop with explicit `break` on exit:

```
while True:
    print("\nDate & Time Menu")
    # ... display options ...
    choice = input("Enter your choice (1-4): ")

    if choice == "4":
        print("Goodbye!")
        break
    elif choice == "3":
        app = wx.App()
        frame = CountdownFrame()
        frame.Show()
        app.MainLoop()  # Run GUI event loop separately
```

This ensures:

- Program continues running until explicit exit
- Invalid choices are caught and user is re-prompted
- wxPython GUI runs in its own event loop without blocking console menu
- User experience is smooth and intuitive

---

# Challenge 5: Date Format Consistency and wxPython Integration

**Problem:**

- Users might expect different date formats (DD/MM/YYYY, MM-DD-YYYY, etc.), leading to confusion or incorrect parsing
- Integrating wxPython GUI with console menu required learning framework basics and event handling

**Solution:**

- Establish a clear, unambiguous format: **YYYY-MM-DD** (ISO 8601 international standard)
- Display the format requirement in every prompt: `"Enter a date (YYYY-MM-DD): "`
- Provide clear error messages if format is incorrect
- ISO 8601 format is sortable, unambiguous across cultures, and widely recognized
- Use wxPython's event binding system to handle button clicks and user interactions without blocking the main thread

**wxPython Implementation:**

```
calculate_btn.Bind(wx.EVT_BUTTON, self.on_calculate)  # Bind event handler
self.result.SetLabel(f"Days left: {delta}")  # Update GUI dynamically
```

---

# Technical Stack Summary

| Component | Technology | Purpose |
|---|---|---|
| Console Interface | Python (input/print) | Menu-driven navigation |
| Date Calculations | datetime module | Accurate date operations |
| GUI Framework | wxPython | Countdown calculator graphical interface |
| Error Handling | try-except blocks | Robust input validation |
| Platform | Cross-platform | Works on Windows, Mac, Linux |

---

# Conclusion

This project successfully demonstrates how Python's `datetime` module can be leveraged to create practical date/time applications, enhanced with wxPython for a modern graphical user interface. The menu-driven architecture makes it scalable—additional features (time zone conversion, holiday calculations, etc.) can be easily added as new functions.

The combination of modular design, robust error handling, Python's powerful standard library, and wxPython's GUI capabilities creates a reliable, user-friendly tool that is both computationally accurate and visually appealing. The GUI implementation for the countdown feature showcases the versatility of Python in creating both console and graphical applications.

## Key Learning Outcomes

- Proficiency with Python's `datetime` library
- Implementation of tuple comparison logic
- Error handling with try-except blocks
- Menu-driven program design
- Input validation and user-friendly error messages
- **wxPython GUI framework fundamentals**
- **Event-driven programming and event handling**
- **Cross-platform GUI application development**
- **Integration of GUI components with console applications**

## Future Enhancements

- Add time zone support for global applications
- Implement holiday calendars for event planning
- Create persistent user preferences storage
- Develop mobile app version using Python mobile frameworks

- Add recurring event countdown functionality
- Implement calendar visualization in GUI